Web程序员成功之路

Python Web



- 迅速提高读者Web开发能力，全面挖掘读者开发潜力
- 一线资深Web程序员经验力作，窗内网独家推荐自学教材
- 15个小时视频教学，简化学习过程
- 60个实战案例与理论知识综合讲解，提高应用能力
- 网站互动教学(www.itzcn.com)，QQ群在线帮助读者解疑

李　勇　王文强　编著

随书附赠 DVD-ROM

清华大学出版社

Web 程序员成功之路

Python Web开发学习实录

李　勇　王文强　编著

清华大学出版社
北　京

内 容 简 介

Python 是目前流行的动态脚本语言之一。

本书共 15 章，由浅入深、全面系统地介绍了使用 Python 语言进行程序开发的知识和技巧。内容包括 Python 的安装和环境配置、Python 的基本语法、流程控制、模块和函数、数据结构、字符串与正则表达式、面向对象编程、文件处理、程序异常和处理、数据库连接和持久化操作、Python 网络功能、Python 与 HTML、XML 的应用、Python 图像界面的处理、Python 的 Web 开发等。

本书适合 Python 爱好者、大中专院校的学生、社会培训班的学生以及使用 Python 语言进行系统管理、GUI 开发、Web 开发、数据库编程和网络编程的程序员使用。

图书在版编目(CIP)数据

Python Web 开发学习实录/李勇，王文强编著. --北京：清华大学出版社，2011.10
(Web 程序员成功之路)
ISBN 978-7-302-26633-4

Ⅰ. ①P…　Ⅱ. ①李…　②王…　Ⅲ. ①软件工具—程序设计　Ⅳ. ①TP311.56

中国版本图书馆 CIP 数据核字(2011)第 174540 号

责任编辑：张　瑜
装帧设计：杨玉兰
责任校对：王　晖
责任印制：

出版发行：清华大学出版社　　**地　址：**北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：**100084
社 总 机：010-62770175　　**邮　购：**010-62786544
投稿与读者服务：010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：190×260　**印　张：**33.25　**字　数：**803 千字
附 DVD 1 张
版　　次：2011 年 10 月第 1 版　　**印　次：**2011 年 10 月第 1 次印刷
印　　数：1~4000
定　　价：66.00 元

产品编号：

前　言

当前，Python 已经成为流行的程序设计语言之一，被越来越多的人作为首选语言来学习和应用。作为一种解释型的语言，Python 具有高效的数据结构，提供了一种简单但很有效的方式以便进行面向对象编程。Python 高雅的语法、动态的数据类型，以及它的解释器，使其成为大多数平台上应用于各领域的理想的脚本语言以及开发环境。

Python 解释器及其扩展标准库的源码和编译版本，可以从 Python 的 Web 站点 http://www.python.org/以及所有镜像站上免费获得，并且可以自由发布。该站点上还提供了 Python 的一些第三方模块、程序、工具以及附加的文档。

Python 解释器很容易通过 C/C++(或者其他可以由 C 调用的语言)来实现功能和数据结构的扩展。因此 Python 很适于作为定制应用的一种扩展语言。为了使广大读者既能了解 Python 语言的基础知识，又能将 Python 语言应用于特定领域(如 Web 开发)，本书全面地介绍了用 Python 语言进行程序开发的相关知识。学习完本书之后，相信读者能够掌握 Python 语言，并且可以使用 Python 语言进行实际项目的开发。

本书内容

第 1 章　欢迎来到 Python 世界。本章首先介绍 Python 的背景、特性和应用，然后详细介绍如何安装 Python 到本机，接着简单介绍 Python 的解释器和开发工具，最后介绍 Python 程序的保存及运行。

第 2 章　练就扎实的基本功。本章首先介绍 Python 语言的变量声明和使用，然后介绍 Python 的命令、数据类型和表达式，最后简单介绍在 Python 语言中如何为代码行添加注释以及 Python 的运算符。

第 3 章　控制结构。本章首先介绍条件语句的使用，然后详细介绍循环语句以及迭代器的使用，最后介绍循环中的跳转语句和其他语句的用法。

第 4 章　可复用的函数和模块。本章首先介绍 Python 程序的结构设计、模块的创建和如何导入模块，然后介绍在 Python 语言中模块的内置函数，最后介绍如何定义函数和调用函数。

第 5 章　数据结构。本章首先介绍列表的创建和使用，然后介绍元组的创建、访问和操作，最后介绍字典的创建、访问和操作，以及有关序列的常识。

第 6 章　字符串与正则表达式。本章首先介绍字符串的拼接、格式化，然后介绍如何截取字符串、比较字符串、搜索字符串和替换字符串，最后介绍如何转换时间字符串。

第 7 章　面向对象编程。本章首先讲解封装、多态的使用，然后介绍如何创建类和对象，最后介绍类的继承和特性。

第 8 章　基于文件的交互。本章首先介绍如何打开一个文件和创建文件，然后介绍如何对文件进行增、删、改、查的操作，最后介绍如何复制文件、修改文件名称和关闭文件，同时介绍文件的一些内置函数、方法和属性等知识。

第 9 章　构造可容错的应用程序。本章首先介绍如何使用 try ...except 语句、try ...finally 语句捕捉异常，然后介绍如何使用 raise 语句抛出异常，最后介绍 assert 语句的使用方法以及程

序的调试技巧。

第 10 章　持久化的数据。本章首先介绍持久化存储的概念，然后介绍数据库的连接及游标的使用，最后介绍如何使用持久化模块来读写数据以及如何操作嵌入式数据库 SQLite。

第 11 章　让信息自由联通——Python 网络功能。本章首先介绍 TCP/IP 网络互连模式，然后介绍 Socket 的基础知识，并详细地介绍服务器和客户端之间的通信技术，最后介绍如何实现异步通信技术。

第 12 章　构造可兼容的应用程序。本章首先介绍 HTML 的语法规范以及对 URL 字符串的处理，然后介绍如何获取 HTML 文档资源，最后介绍如何解析 HTML 文档。

第 13 章　应知应会技能之 XML 处理。本章首先介绍 XML 文档的结构，然后介绍如何使用 SAX 处理 XML 文档以及如何使用 DOM 解析 XML 数据，最后介绍可扩展样式表语言 XSL。

第 14 章　图形用户界面。本章首先介绍 WxPython 的程序结构，然后介绍 WxPython 的基本组件和常用组件，最后介绍 WxPython 库中的菜单控件。

第 15 章　Python 的 Web 开发之 Django 框架应用。本章首先介绍 Django 框架中的 MVC 架构，然后介绍 Django 框架的开发环境搭建，最后介绍 Django 框架的应用和高级应用。

本书特色

本书的大量内容来自真实 Python 项目，力求通过读者实际操作时的问题解答方式使读者更容易掌握 Python 应用开发。本书难度适中，内容由浅入深，实用性强，覆盖面广，条理清晰。

(1) 结构独特。通过“网络教学、基础知识、实例描述、实例应用、运行结果、实例分析”形式将每个知识点与实际应用中的问题相结合。

(2) 形式新颖。用准确的语言总结概念，用直观的图示演示过程，用详细的注释解释代码，用形象的比方帮助记忆。

(3) 技术文档。将一些非常简单的知识点或者理论性的内容安排在这里。通常这些文档让读者了解有关的概念和术语。

(4) 内容丰富。涵盖了实际开发中 Python 技术所遇到的 XML、HTML 和 Django 等方面的热点问题。

(5) 随书光盘。本书为实例配备了视频教学文件，读者可以通过视频文件更加直观地学习 Python 的使用方法和知识。

(6) 网站技术支持。读者在学习或者工作的过程中，如果遇到实际问题，可以直接登录 www.itzcn.com 与我们取得联系，作者会在第一时间内给予帮助或建议。

(7) 贴心的提示。为了便于读者阅读，全书还穿插了一些技巧、提示等小贴士，并有各自的体例约定。

提示：通常是一些贴心的提醒信息，让读者加深印象或提供建议和解决问题的方法。

注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关信息。

技巧：通过简短的文字，指出知识点在应用时的一些小窍门。

读者对象

本书具有知识全面、实例精彩、指导性强的特点，力求以全面的知识性及丰富的实例来指导读者透彻地学习 Python 各方面的知识。本书可以作为 Python 的入门书籍，也可以帮助中级

读者提高技能，对高级读者也有一定的启发意义。

参编人员

本书主要由李勇、王文强编写，其他参与编写、资料整理、程序开发的人员还有陈军红、郝军启、王俊伟、赵振方、张芳芳、祝红涛、徐牛犇等。

由于编者水平有限，书中难免存在不足和疏漏之处，恳请读者批评指正。

本书适合以下人员阅读学习：

- Python 语言爱好者
- 大中专院校的学生
- 社会各类培训班学生
- 从事各种工作的程序员
- 具有其他编程使用经验的开发人员

目　　录



第 1 章　欢迎来到Python世界

内容摘要

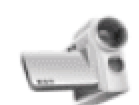
Python 是一种功能强大而完善的通用语言，也是一种直译式计算机程序设计语言。该语言借鉴了简单脚本和解释性语言的易用性，能够高效地完成各种复杂的高层次任务。本章将简单介绍 Python 的知识和特性，重点介绍 Python 运行平台的搭建，以及使用解释器保存并执行程序。

学习目标

- 了解 Python 的背景。
- 了解 Python 的特性和应用。
- 熟练掌握如何安装 Python。
- 了解 Python 解释器。
- 了解使用 Python 语言的开发工具。
- 掌握 Python 程序的保存及执行。

1.1　Python简介

Python 语言的名字来源于一个喜剧，它的设计在工业和科研上得到了广泛的应用，被誉为成为黑客应学习的四大编程语言之一。


视频教学：光盘/videos/01/python 简介.avi　　　　　**长度：**11 分钟

Python 是一种功能强大但简单易学的语言，本节我们将介绍什么是 Python 以及 Python 的背景、特性、缺点和应用。

1. 什么是Python

Python 语言是一种功能非常强大的编程语言，它既具有简单脚本和解释型语言的易用性，也具有传统编译语言的强大性，其高效率的高层数据结构和简单易学的面向对象编程特点，在很多平台上都得到了认可，成为在这些平台上进行快速应用程序开发的主流脚本语言。

2. Python的背景

Python 是在一个名为 Guido van Rossum 的人手中诞生的，设计之初的理念是为了让设计人员不用为编程语言的结构而烦恼，使程序员能够专心于实现程序的功能。

1989 年圣诞节期间，Guido 对解释型语言 ABC 有着丰富的设计经验，但是 ABC 语言由于没有开源，造成了它的不成功。Guido 决心开发一个新的脚本解释程序，作为 ABC 语言的一种继承，实现 ABC 语言未能实现的东西，就这样 Python 语言诞生了。

3. Python语言的特性

Python 语言有很多特性，它使得编写程序变得更为简单有趣，并成为非常精彩而强大的语言，下面我们来看看它的特性。

1) 免费开源

像大家都知道的 Java、PHP 等语言都是开放源代码的，这些语言都得到了广大编程人员的认可，而 Python 也是考虑到长远的发展，采取了向公众开放源代码。这样就能使任何一个爱好者都能够自由发布这个软件的拷贝、阅读源代码并把它运用到新的开源软件中。正因如此，Python 一直被一些更加优秀的人不断改进着。

2) 简单易学

Python 具有很强的伪代码特性，我们在阅读源代码时，就像在读英语文章一样，学习起来很容易，正是因为 Guido 的这种设计理念，使得程序员只专注于解决问题而不是如何搞明白语言本身。

3) 高级语言

Python 是一种高级语言，不像使用汇编语言等那样要考虑诸如如何管理你的程序所使用的内存之类的底层细节。

4) 解释执行

Python 是一种解释型的语言，使用这种语言编写的程序，不需要编译成计算机认可的二进

制代码，而是直接从源代码运行程序。

在计算机内部，像使用 C/C++等编译型语言编写的程序，必须通过编译器和不同的标记、选项把程序的源代码编译成计算机可识别的二进制语言。当运行程序时，连接/转载器软件再把程序从硬盘复制到内存中并且执行。而 Python 程序是通过 Python 解释器解释并执行的，Python 解释器把程序的源代码转换成称为字节码的中间形式，然后再把它翻译成计算机语言并执行，使得程序员无须关心程序如何编译、程序中用到的库如何加载等麻烦问题。这样，使用 Python 将会更加简单，也更容易移植。

5) 可移植性

Python 具有强大的可移植性，而且移植起来也很简单，只需要把 Python 程序拷贝到另一台计算机上就行了。由于 Python 是开放源代码的语言，它已经被移植到了许多平台上，甚至 Linux 系统中还内置了 Python。如果你小心地避免使用依赖于系统的特性，那么你编写的任何 Python 程序都可无须修改地在表 1-1 所示的现今主流系统平台上运行。

表 1-1　运行Python的主流平台

Linux	Windows	FreeBSD	Macintosh	Solaris	OS/2
Amiga	AROS	AS/400	BeOS	OS/390	z/OS
Palm OS	QNX	VMS	Psion	Acom RISC OS	VxWorks
PlayStation	Sharp Zaurus	Windows CE	PocketPC	—	—

6) 面向对象

像 Java 语言一样，Python 也支持面向对象编程，不同的是它还支持面向过程的编程。在面向对象的编程中，程序是由数据和功能组合而成的对象构建起来的。而在面向过程的编程中，程序是由过程或可重用的代码函数来构建起来的。就 Python 的面向对象编程而言，Python 还有世界上最强大的类库，我们可以一种简单的方式来实现面向对象编程。

7) 嵌入性

Python 的嵌入性是指它可以作为一种成熟的脚本语言，并且以一种很方便的方式嵌入到其他程序。比如 C/C++中。

8) 可扩展性

Python 的可扩展性使得程序员能够灵活地附加程序和定制工具，缩短开发周期。因为 Python 是基于 C 语言开发的，所以用 C/C++来编写 Python 的扩展。但是发展到现在 Python 也有支持 Java 实现的 Jpython 扩展，从而使得 Python 可以在更多的语言中使用。

9) 丰富的类库

Python 是世界上具有标准库最大的编程语言。基于庞大的标准库，我们可以编写程序来处理各种工作。其类库实现的主要功能如表 1-2 所示。

表 1-2　类库实现的功能

正则表达式	文档生成	单元测试	线　程
数据库	网页浏览器	CGI	FTP
电子邮件	XML	XML-RPC	HTML
WAV 文件	密码系统	GUI(图形用户界面)	Tk

10) 内存管理器

在程序开发过程中，我们会遇到像使用 C/C++时要考虑的程序的内存管理问题。即使你使用的是很小的程序，应用程序的修改和管理也需要程序员额外负责，这就需要开发者付出更多的精力。

而在 Python 的程序开发过程中，Python 解释器承担了程序的内存管理工作，使得程序员从内存事务处理中解脱出来，全身心致力于程序功能的实现，从而减少错误，健壮程序，缩短开发周期。

4. Python的应用

自 2006 年以来，Python 已成为继 C++、Java 之后的第三种编程语言，更多地被应用到著名搜索引擎 Google 和 Nokia 智能手机所采用的 Symbian 操作系统上，可见 Python 的应用领域非常广泛。表 1-3 介绍了 Python 语言可能的应用范围。

表 1-3　Python的应用范围

应用范围	详细描述
系统编程	提供 API，能方便地进行系统维护和管理，是很多系统管理员理想的编程工具，是 Linux 系统下的标志性语言之一
图形处理	含有庞大的对诸如 PIL、Tkinter 等图形类库的支持，能够方便地进行图形处理
数字处理	NumPy 扩展提供大量与许多标准数学库对应的接口，可以方便地处理数学问题
文本处理	Python 提供了很多模块，如 re 模块能够处理正则表达式，又如 SGML、XML 分析模块可进行文本的编程开发
数据库编程	通过 Python DB-API(数据库应用程序编程接口)规范模块，可以与 Microsoft SQL Server、Oracle、Sybase、DB2、Mysql、SQLite 等数据库通信。Python 自带的 Gadfly 模块可提供完整的 SQL 环境
网络编程	提供丰富的模块支持 Sockets 编程，能够方便、快速地开发分布式应用程序
Web 编程	支持 HTML、XML 技术
多媒体应用	Python 的 PyGame 模块可用于编写游戏软件，yOpenGL 模块则封装了 OpenGL 应用程序编程接口，能进行二维和三维图像处理

1.2　安装Python(Windows安装)

Python 是一种功能非常强大的语言，通过学习，我们很快就能完成与平台无关的各种程序。在了解了 Python 的特点之后，我们就可以进行 Python 语言开发的学习了。在学习之前，首先要了解 Python 的下载、安装以及怎样启动。

视频教学：光盘/videos/01/ Python 的安装.avi　　　长度：8 分钟

用 Python 编写的程序只有在安装了 Python 和配置好环境的前提下才能运行，在这里我们将讲解如何下载、安装和启动 Python。值得一提的是，大约 70%的下载都来自 Windows 用户，

不要以为这样就可以下结论说 Python 在 Windows 平台中使用得最多，因为几乎所有的 Linux 用户在安装系统时都默认已经安装了 Python。

我们可以从 Python 的官方网站下载该软件。打开浏览器，在地址栏输入 http://www.python.org/ 打开网站主页，然后下载安装文件。

1. Python的安装

(1) 在目录中找到刚刚下载的 Python 安装文件，双击这个文件，会弹出 Python 安装程序的安装向导对话框，如图 1-1 所示。

(2) 在这里我们可以看到两个单选按钮，第一个 Install for all users 是为所有用户安装，第二个 Install just for me 是为个人用户安装。单击第一个按钮，然后单击 Next 按钮，进入 Python 的安装路径设置界面，如图 1-2 所示。

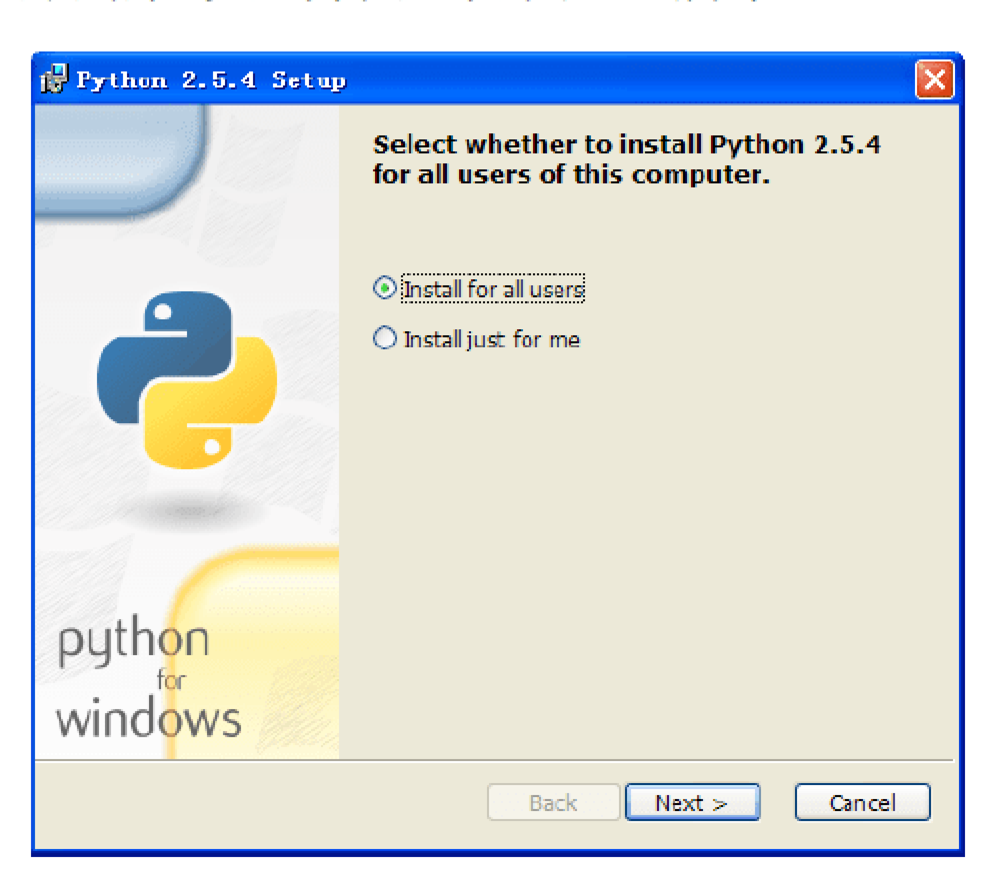


图 1-1　Python安装程序向导对话框

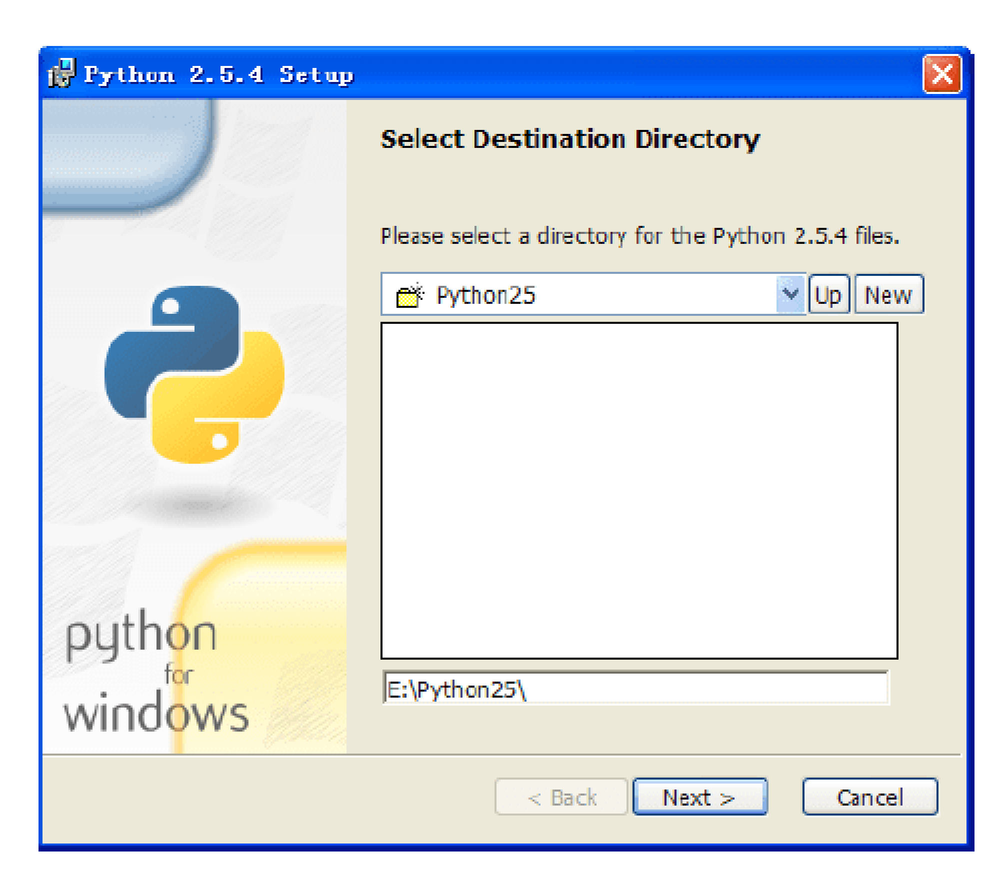


图 1-2　Python安装路径设置界面

(3) 选择安装路径。可以把路径更改为硬盘的任意路径。在此我们把 Python 安装到 E:\Python25 下，选择好安装路径后，单击 Next 按钮，进入 Python 安装组件选择界面。在这里我们选择安装所有的组件，一般采用默认就可以了，如图 1-3 所示。

(4) 单击 Next 按钮，进入 Python 工具包的安装界面，如图 1-4 所示。



图 1-3　Python组件选择界面

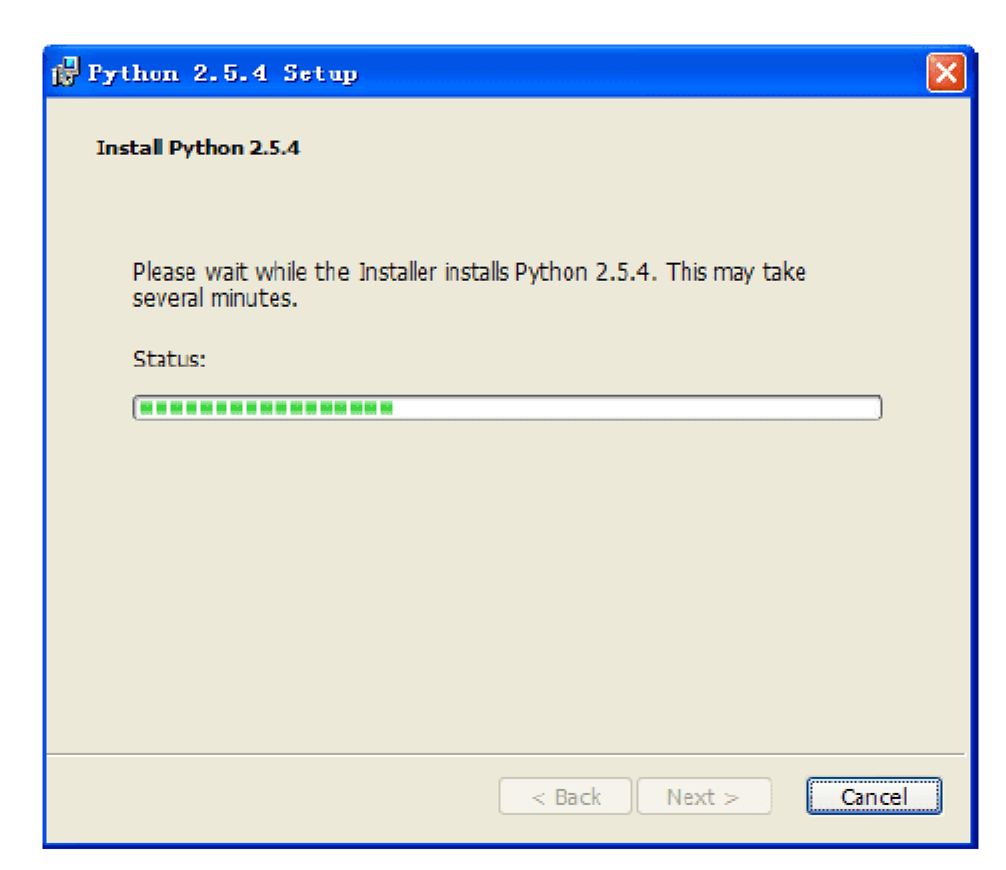


图 1-4　Python工具包安装界面

(5) 等程序安装完成后，会提示如图 1-5 所示的程序安装成功界面，最后单击 Finish 按钮完成安装。



图 1-5　Python安装完成

2．Python的环境配置和测试

我们已经安装好了 Python，但是想要正常运行它还差一步，那就是 Python 的环境配置。在电脑的桌面上右击“我的电脑”图标，在弹出的快捷菜单中选择“属性”命令，会出现“系统属性”对话框，然后切换到“高级”选项卡，如图 1-6 所示，会出现“环境变量”按钮。

单击“环境变量”按钮，出现系统所有的环境变量列表，如图 1-7 所示。

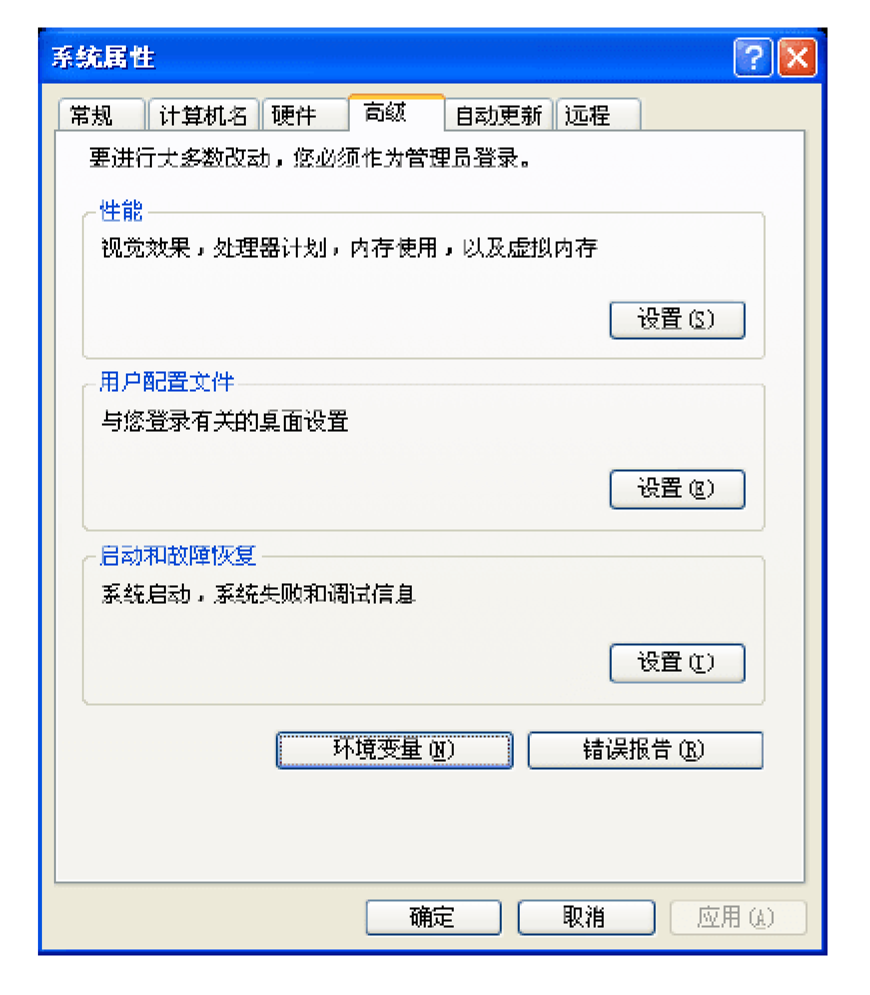






图 1-6　“系统属性”对话框　　　　图 1-7　系统环境变量

从“系统变量”列表框中找到名为 Path 的变量，选中并单击“编辑”按钮，出现“编辑系统变量”对话框，如图 1-8 所示。在“变量值”文本框中添加已经安装的 Python 路径 E:\Python25，然后以分号隔开，单击“确定”按钮。这样就完成了 Python 的环境配置。

当正确安装了 Python，并配置了 Python 环境变量后，我们就可以正常运行 Python 了。在这里我们可以通过两种方式来启动 Python：一种是使用命令行启动，另一种是使用 Python 的

集成开发环境 IDLE。

1．Python的命令行启动

执行“开始”｜“运行”命令，打开“运行”对话框。在该对话框中输入 python 命令，如图 1-9 所示。

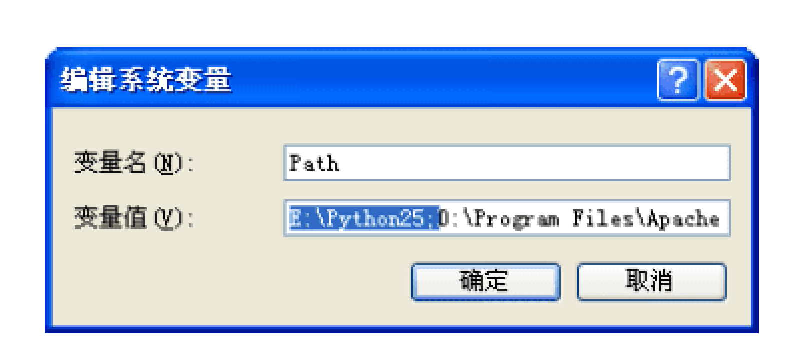


图 1-8　编辑环境变量

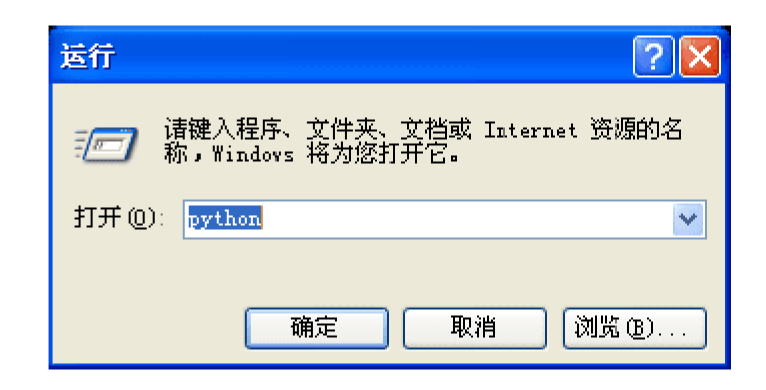


图 1-9　“运行”对话框

直接回车或单击“确定”按钮，就会在 DOS 环境中正常启动 Python 了，如图 1-10 所示。

```
D:\Program Files\Python\python.exe
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on
win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

图 1-10　在DOS下启动Python

2．使用Python集成开发环境启动

除了使用命令行启动 Python 外，我们还可以使用其他方式来启动 Python，例如使用 IDLE 集成开发工具。执行“开始”｜“程序”｜Python 2.5｜IDLE(Python GUI)命令，启动 Python 集成开发环境，如图 1-11 所示。

```
Python Shell
File  Edit  Shell  Debug  Options  Windows  Help
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on
win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.4
>>> |
Ln: 12 Col: 4
```

图 1-11　IDLE交互式Python Shell

1.3　使用带提示符的解释器

Python 是一种解释型的语言，并不需要编译而是直接在机器上执行。在编写 Python 程序时，我们需要了解 Python 解释器的有关知识。本节将讲解如何调用解释器、解释器的错误处理、源程序编码和交互式环境的启动文件等内容。

视频教学：光盘/videos/01/Python 解释器.avi　　　　长度：10 分钟

1.3.1　基础知识——Python解释器

除了 Python 的语言特色外，Python 解释器就是整个语言能够得以运行的灵魂。有了 Python 解释器，用 Python 编写的程序才能处理相应的事务。

1．调用解释器

在使用 Python 编程的时候，要用到 Python 解释器来解释并执行程序。在 Windows 平台下启动解释器有两种方式：一是在命令行中启动，二是使用 IDLE。下面分别看看这两种启动方式。

1) 在命令行启动解释器

如果像 1.2 节所讲的，我们应经配置好了 Python 的运行环境，这样我们就可以在命令行启动解释器了。

首先打开 DOS 命令提示符窗口，如图 1-12 所示。

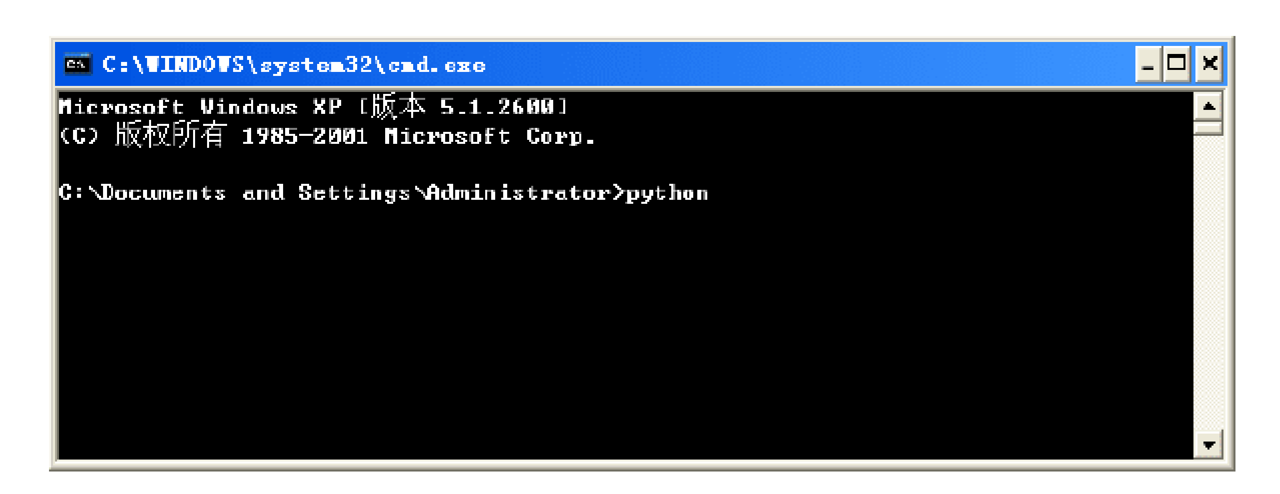


图 1-12　DOS命令提示符窗口

在该窗口中输入 python 命令并回车，就能顺利启动 Python 解释器，如图 1-13 所示。该解释器会提示有关解释器的版本、时间和系统平台等信息。

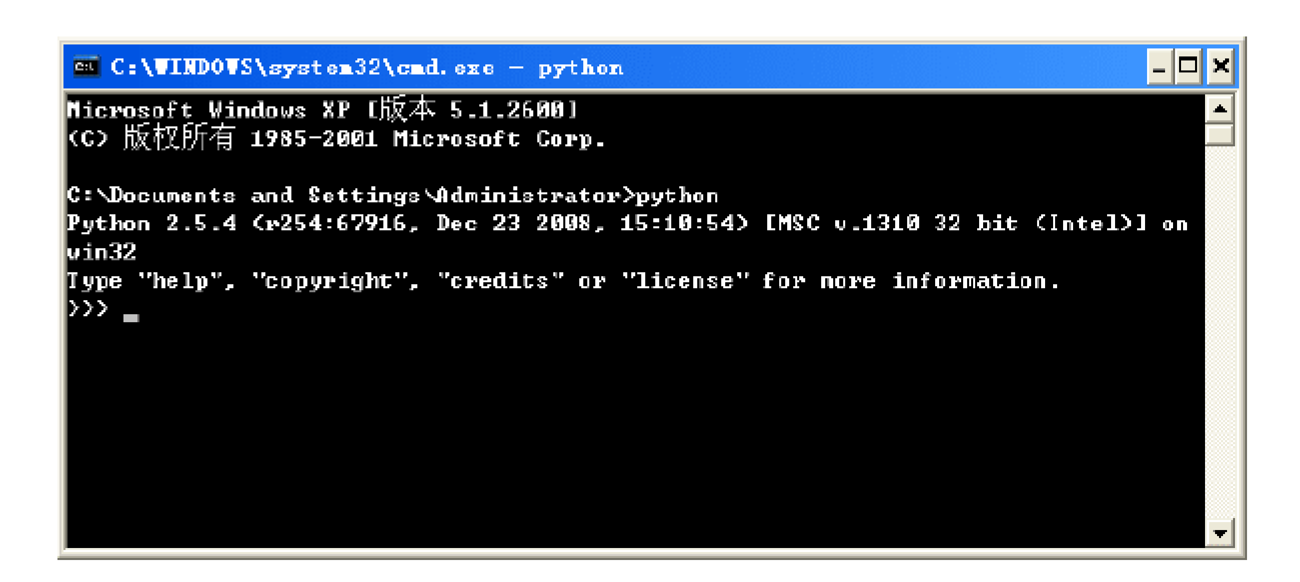


图 1-13　用DOS命令启动解释器

通过命令行启动解释器后，我们可以通过执行有关命令让解释器执行一些动作，如表 1-4 所示。

表 1-4　解释器命令

命　令	作　用
-d	提供调试输出
-O	生成优化的字节码(生成.pyo 文件)
-S	不导入 site 模块，以在启动时自动查找 Python 路径
-v	冗余输出(导入语句详细追踪)
-m mod	将一个模块以脚本形式运行
-Q opt	除法选项
-c cmd	运行以命令行字符串形式提交的 Python 脚本
file	从给定的文件运行 Python 脚本

2) 使用 IDLE 启动解释器

IDLE 是用于 Python 程序开发的集成开发工具，通过它同样可以启动 Python 解释器。如前所述，可以执行“开始”|“程序”| Python 2.5 | IDLE(Python GUI)命令来启动 Python 集成开发环境，如图 1-14 所示。

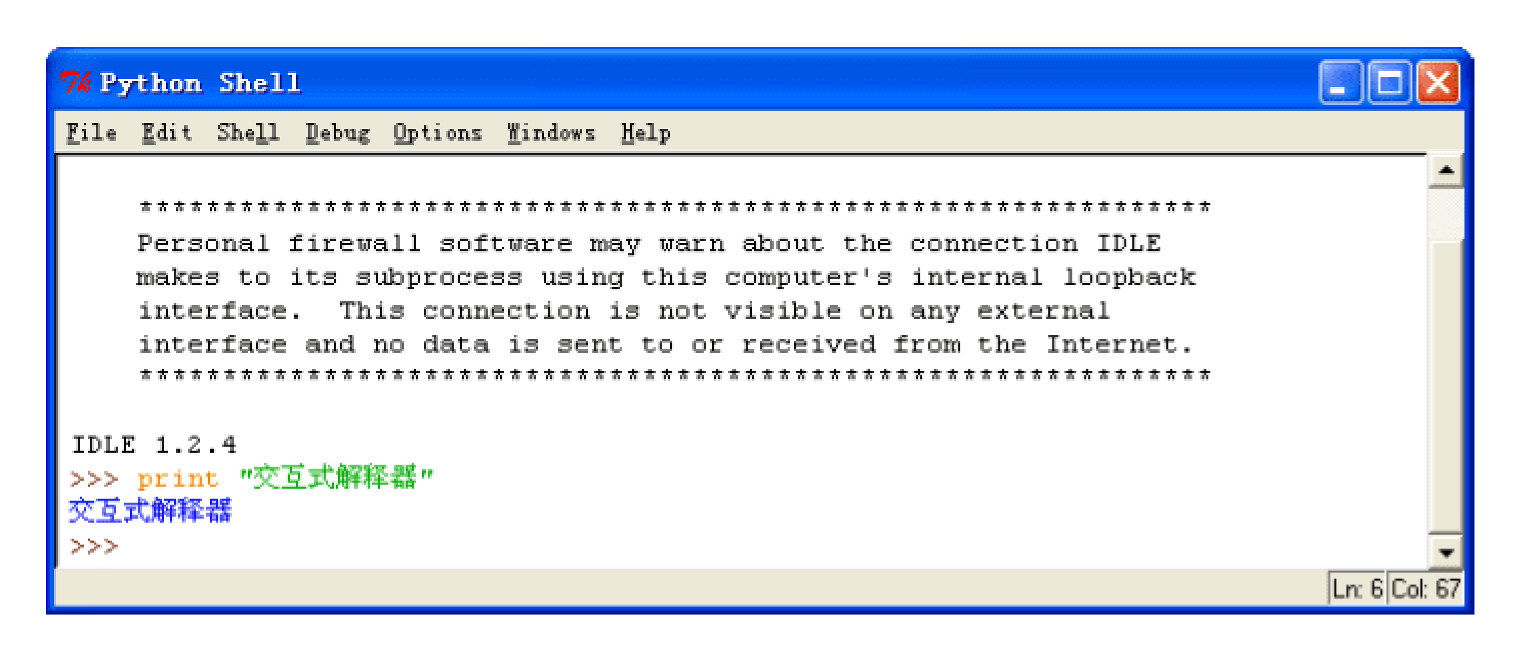


图 1-14　IDLE

在图 1-14 中，执行 File | new Window 命令，或者使用快捷键 Ctrl+N，打开一个新的编辑窗口。在里面，我们输入以下代码，这就是在编写代码了，如图 1-15 所示。

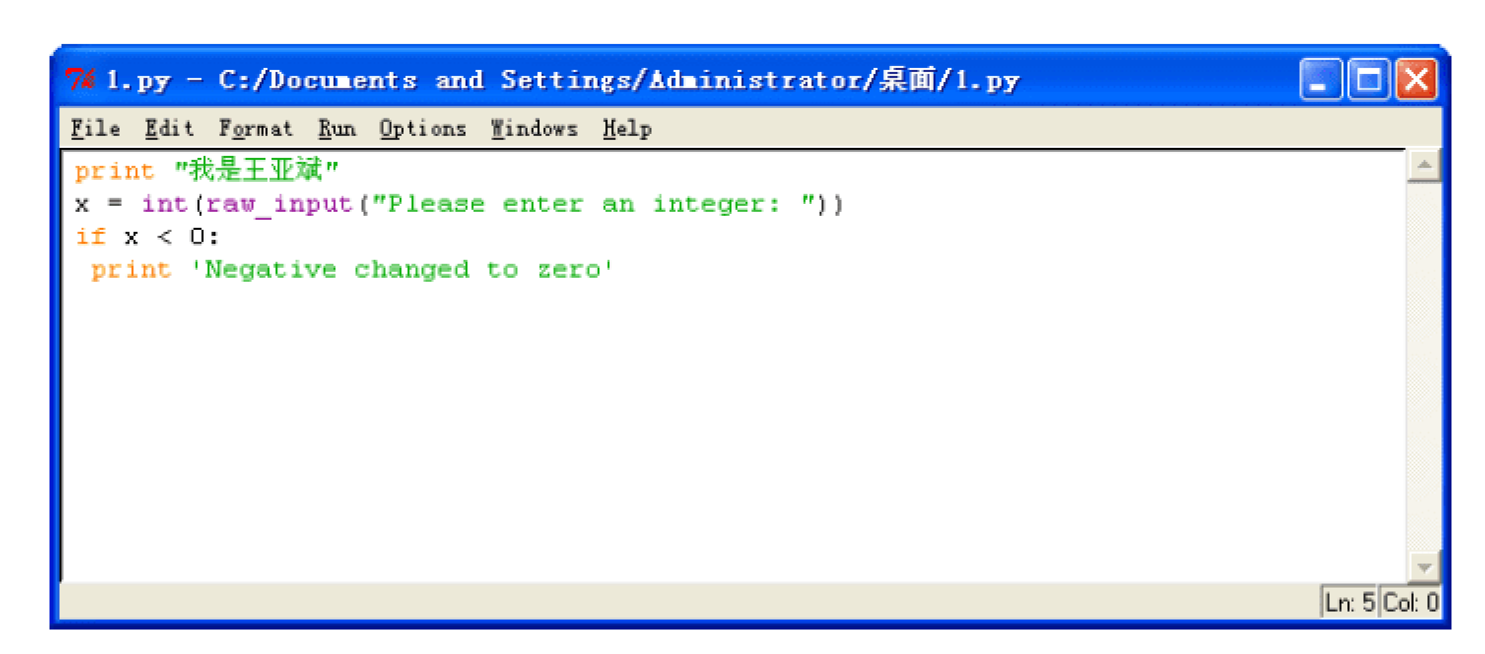


图 1-15　在编辑器中编辑程序

```
print "我是王亚斌"
x = int(raw_input("Please enter an integer: "))
if x < 0:
 print 'Negative changed to zero'
```

把编辑好的程序保存，然后使用快捷键 Ctrl+F5，将程序交给 Python 解释器解释执行，其执行结果如图 1-16 所示。

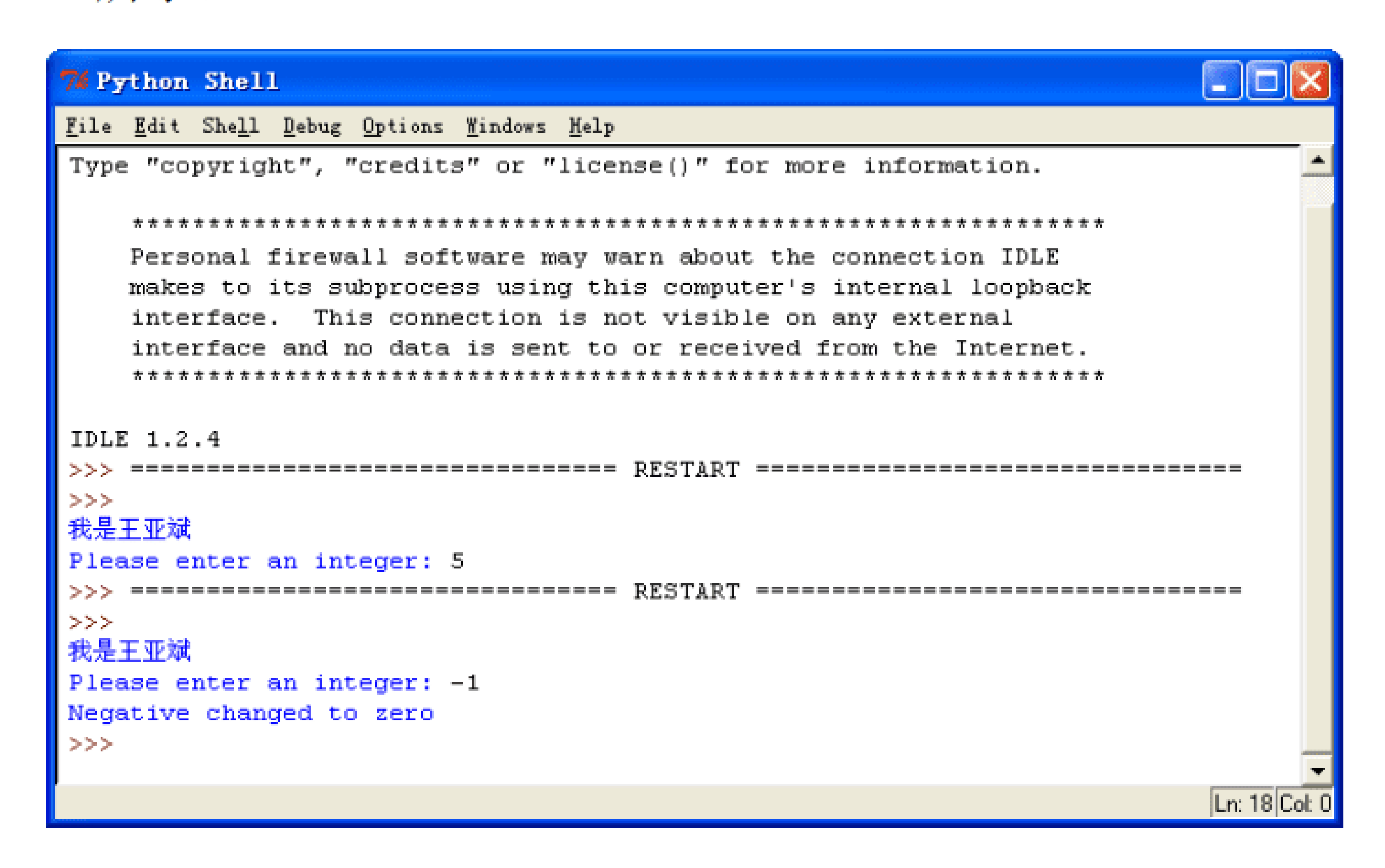


图 1-16　程序执行结果

2．交互模式

在 Python 解释器中，我们编写的程序可以交互式模式解释执行。在这种模式下它主要根据主提示符来执行，Python 中的主提示符标记通常是 3 个大于号(>>>)，除此之外，还有从属提示符，以 3 个点来标记(...)。启动解释器后，我们将看到解释器打印的欢迎信息、版本号和授权提示信息，如下所示：

```
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
```

当 Python 解释器解释程序时，会在下一行立即给出程序的结果，使结果和程序进行交互式执行。

3．解释器的错误处理

在编写程序时，难免遇到这样或那样的错误，在 Python 中也会发生。当有错误发生时，解释器会打印错误信息并启用栈跟踪器。在交互模式下，它返回主提示符，如果从文件输入执行，它在打印栈跟踪器后会以非零状态退出。有一些致命的错误会导致系统在非零状态下退出，这些错误通常是由内部矛盾和内存溢出所造成的。

4．源程序编码

在 Python 的源程序中，我们除了使用 ASCII 编码外，还可以使用其他编码方式。具体的做法就是在#!行后面用一个特殊的注释行来定义字符集，例如以下代码：

```
# -*- coding: iso-8859-1 -*-
```

根据这个声明，Python 就会将文件中的字符尽可能地从指定的编码转换为 Unicode。如果文本编辑器支持 UTF-8 格式，也可以转换为此格式。

使用 UTF-8 编码(无论是用标记还是编码声明)，我们可以在字符串和注释中使用世界上的大部分语言。标识符中不能使用非 ASCII 字符集。为了正确显示所有的字符，一定要在编辑器中将文件保存为 UTF-8 格式，而且使用支持文件中所有字符的字。

5. 交互式环境的启动文件

编写程序时，我们会发现大量的代码都可以重用。但在使用 Python 解释器的时候，我们可能在每次启动解释器时执行一些重复的命令，因此我们可以在一个文件中写入这些命令，然后在环境变量中指定名为 PYTHONSTARTUP 的环境变量来指定这个文件，该文件就是交互式环境的启动文件。

在使用这个文件的时候，具体来说就是在交互会话期是只读的，但是当 Python 解释器从脚本解释文件或以终端作为外部命令源时就不会如此。因为它与解释器执行的命令处在同一个命名空间，所以由它定义或引用的一切附加文件都可以在解释器中不受限制地使用。

1.3.2 实例描述

我们知道，Python 解释器在启动时都会执行一些重复的命令，这些命令可以统一写入交互式环境的启动文件中。下面以一个小实例来演示如何执行这个文件。

1.3.3 实例应用

【例 1-1】在当前目录中执行附加的启动文件。

(1) 新建一个名为 too.py 的全局启动文件，放在 E:\Python25 目录下，然后在文件中加入以下代码：

```
if os.path.isfile('.pythonrc.py'): execfile('.pythonrc.py')
```

(2) 打开系统的环境变量，在系统环境变量中重新编辑一个名为 PYTHONSTARTUP 的环境变量，变量值为 E:\Python25\too.py，如图 1-17 所示。

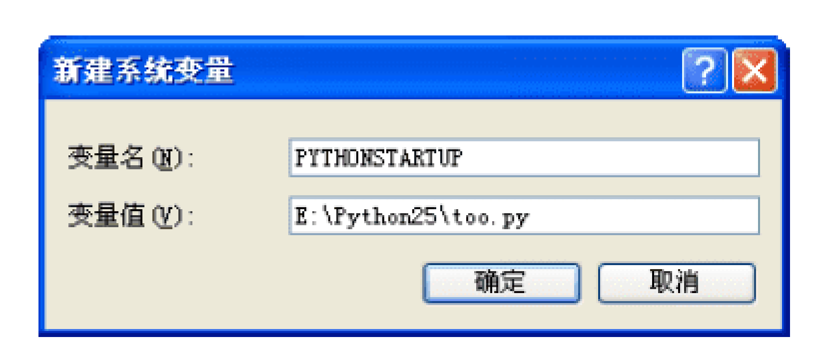


图 1-17 编辑全局启动文件环境变量

(3) 设置好后就能使用了。使用时，我们需要在所使用的文件中加入一些代码。例如新建一个 1.3.py 文件，并在其中加入如下代码：

```
import os
filename = os.environ.get('PYTHONSTARTUP')
if filename and os.path.isfile(filename):
```

```
 execfile(filename)
print "这是引入的交互式启动文件"
```

(4) 保存文件，这样就能在 Python 解释器执行时启动这个文件。

1.3.4 运行结果

选中 1.3.py 文件，然后使用快捷键 Ctrl+F5 使 Python 解释器解释这个文件，结果如图 1-18 所示。

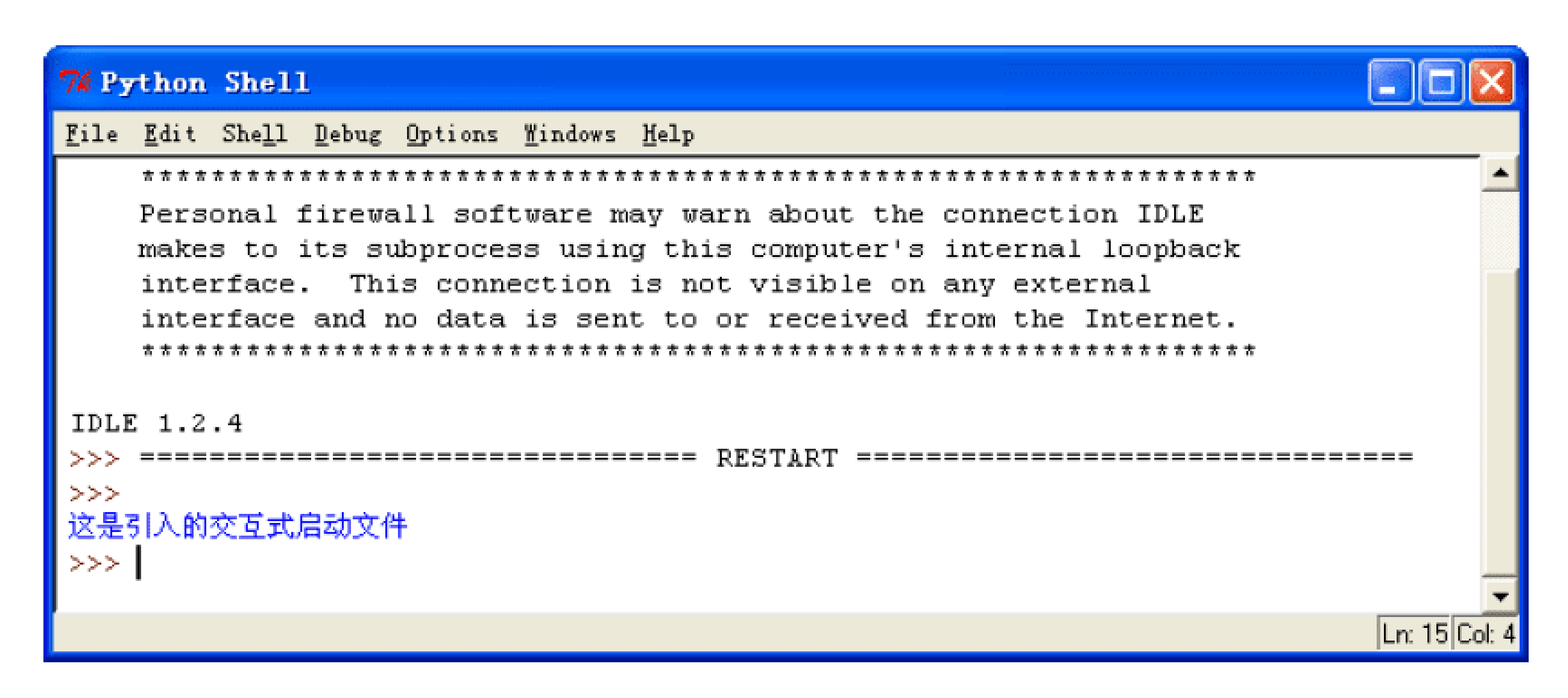


图 1-18 执行交互式启动文件

1.3.5 实例分析


源码解析

在上述实例中，我们定义了一个名为 PYTHONSTARTUP 的环境变量，之后才能在文件中正常启动交互式启动文件。在文件中使用的时候，我们必须使用 import 关键字引入这个文件。

1.4 Python集成开发环境

在 Python 程序的开发中，我们能够看到 Python 的开发工具很多，例如很多强大的智能开发环境(IDE)，像 PythonWin、Eclipse 和 Komodo 等。这些开发环境不仅支持图形化操作，还具有编辑、调试和纠错功能。除了集成开发环境之外，我们经常使用的文本编辑器也可用于 Python 程序的开发，例如 EditPlus 等。


视频教学：光盘/videos/01/ Python 的开发工具.avi　　长度：14 分钟

虽然 Python 的开发工具很多，但是比较常用的只有几个，例如 PythonWin、Eclipse 和 EditPlus。表 1-5 列出了除 Python 官方版本的 IDLE 外的其他集成开发环境。

表 1-5　Python的集成开发环境

集成开发环境	详细描述
IDLE	标准 Python 环境
PythonWin	面向 Windows 的环境
ActivePython	功能完善，包含 PythonWin
Komodo	商业化 IDE
WingWear	商业化 IDE
BlackAdder	商业化 IDE 以及 GUI 生成器
Boa constructor	免费的 IDE 和 GUI 生成器
Anjuta	Linux 和 UNIX 下的万能生成器
Arachno Python	商业化 IDE
Eclipse	流行、灵活并且开源的 IDE
WxGlade	免费的 GUI 生成器
KDevelop	针对 KDE 多语言的 IDE

这里我们讲解一下 PythonWin 的使用方法、Eclipse IDE 集成开发环境以及 EditPlus 编辑器环境的配置。

1．PythonWin的使用方法

PythonWin 的发行版本包括 Windows 应用程序接口和 COM 组件模型，它是世界上最早出现的 Python 开发工具之一。当我们从官方网站下载并安装完成后，就可以通过执行“开始”|“程序”|ActiveState ActivePython 2.5|PythonWin Editor 命令来运行 PythonWin，打开 PythonWin 的图形化命令窗口，如图 1-19 所示。

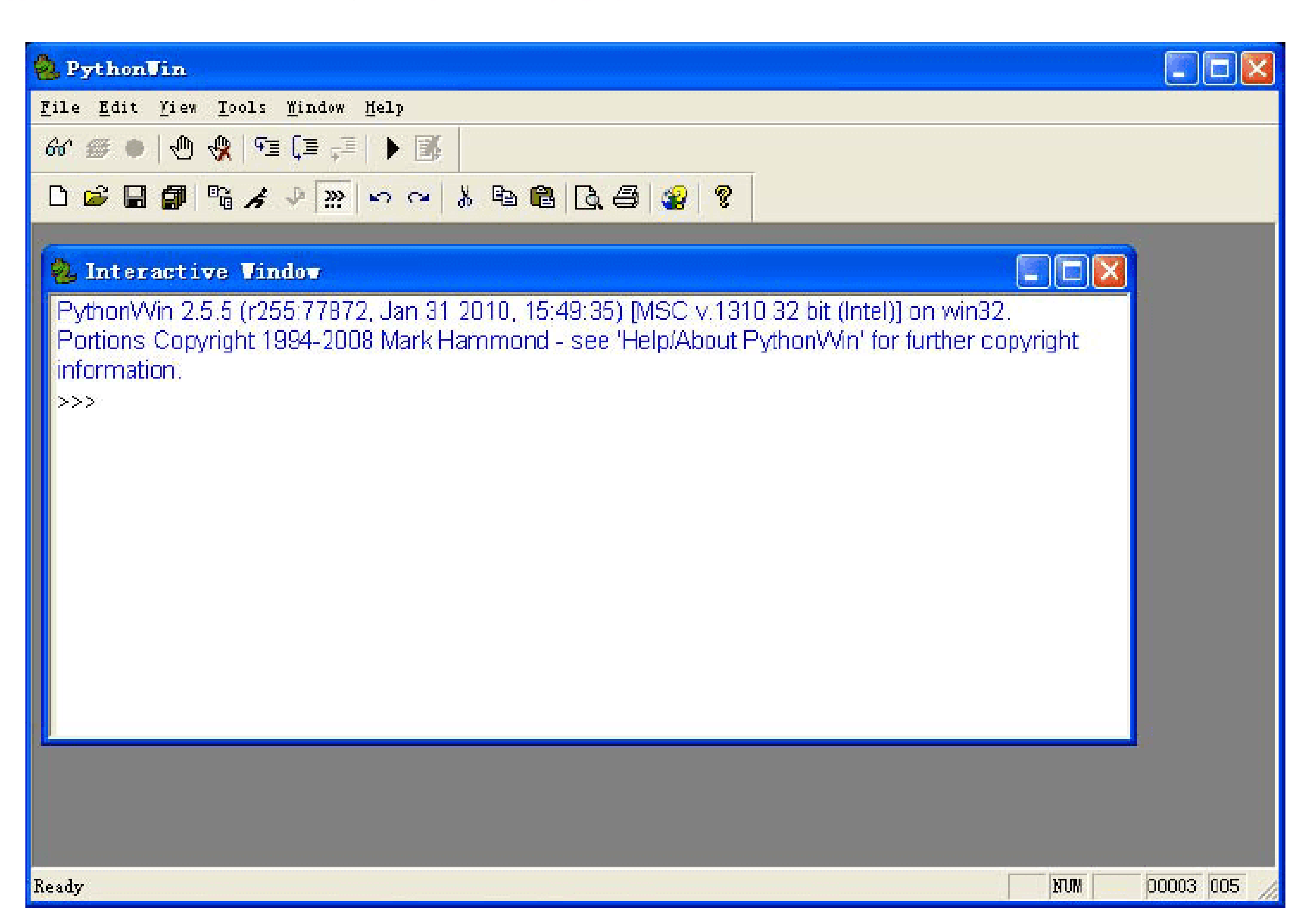


图 1-19　PythonWin图形化命令窗口

在 PythonWin 图形化命令窗口中，通过执行 File | Open 命令来打开文件，并在窗口中运行

文件。要新建一个文件，可以通过执行 File | New 命令，然后在弹出的对话框中选择 Python Script 并确定，这样就可以新建一个 Python 文件并编写 Python 代码了，如图 1-20 所示。

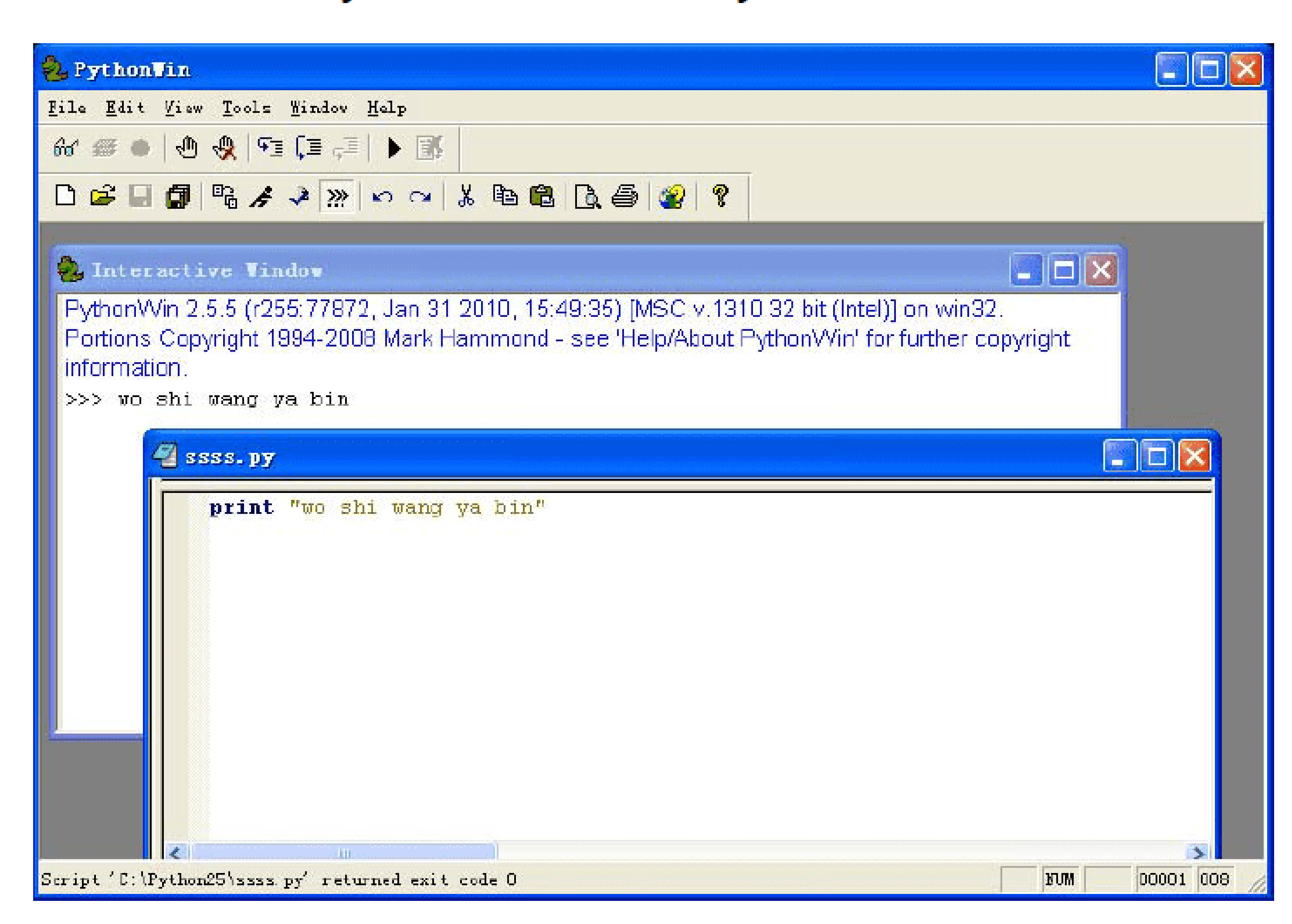


图 1-20　PythonWin编辑器

当使用 PythonWin 做 Python 程序开发时，不仅可以编辑程序，还可以对程序进行断点的设置并进行单步调试，这样就大大降低了程序的出错率。

2. Eclipse IDE集成开发环境

Eclipse 是一个开源项目，它是 Java 开发的集成开发环境。不仅用于 Java，Eclipse 的扩展性也很强，而且能够开发大量的插件来支持其他语言，例如 C、C++、PHP 和 Python 等。

Eclipse 的功能非常强大，它不仅实现了 Python 代码的语法加亮、代码提示和代码补全等智能化功能，而且还提供了比 PythonWin 更强大的调试能力。Eclipse 还支持 Jython、Pyunit 以及团队开发等其他功能。

如果要在 Eclipse 平台上开发 Python，需要下载 PyDev 插件，这里我们使用 Eclipse 对 Python 的独立运行版本 Eclipse for Python，这个工具可以在网上下载。

Eclipse 开发工具的界面主要被分为视图和编辑器两部分，视图部分包括源代码大纲视图和文件系统导航视图。编辑器部分主要包括 Java 源代码编辑器和 Python 源代码编辑器。其视图界面如图 1-21 所示。

3. EditPlus编辑器环境的配置

在开发中，除了使用开发工具外，还可以使用编辑器进行开发。最常使用的编辑器就是 EditPlus。使用 EditPlus 进行程序开发不仅使编写的程序具备语法加亮、代码自动缩进等功能，还可以对程序进行调试。下面介绍一下 EditPlus 编辑器环境的配置。

1) 向 EditPlus 中添加 Python

当我们启动 EditPlus 后，从菜单栏选择“工具”|“配置用户工具”命令，打开“参数选择”对话框。在“参数选择”对话框中单击“添加工具”按钮，再从弹出的菜单中选择“应用程序”命令。

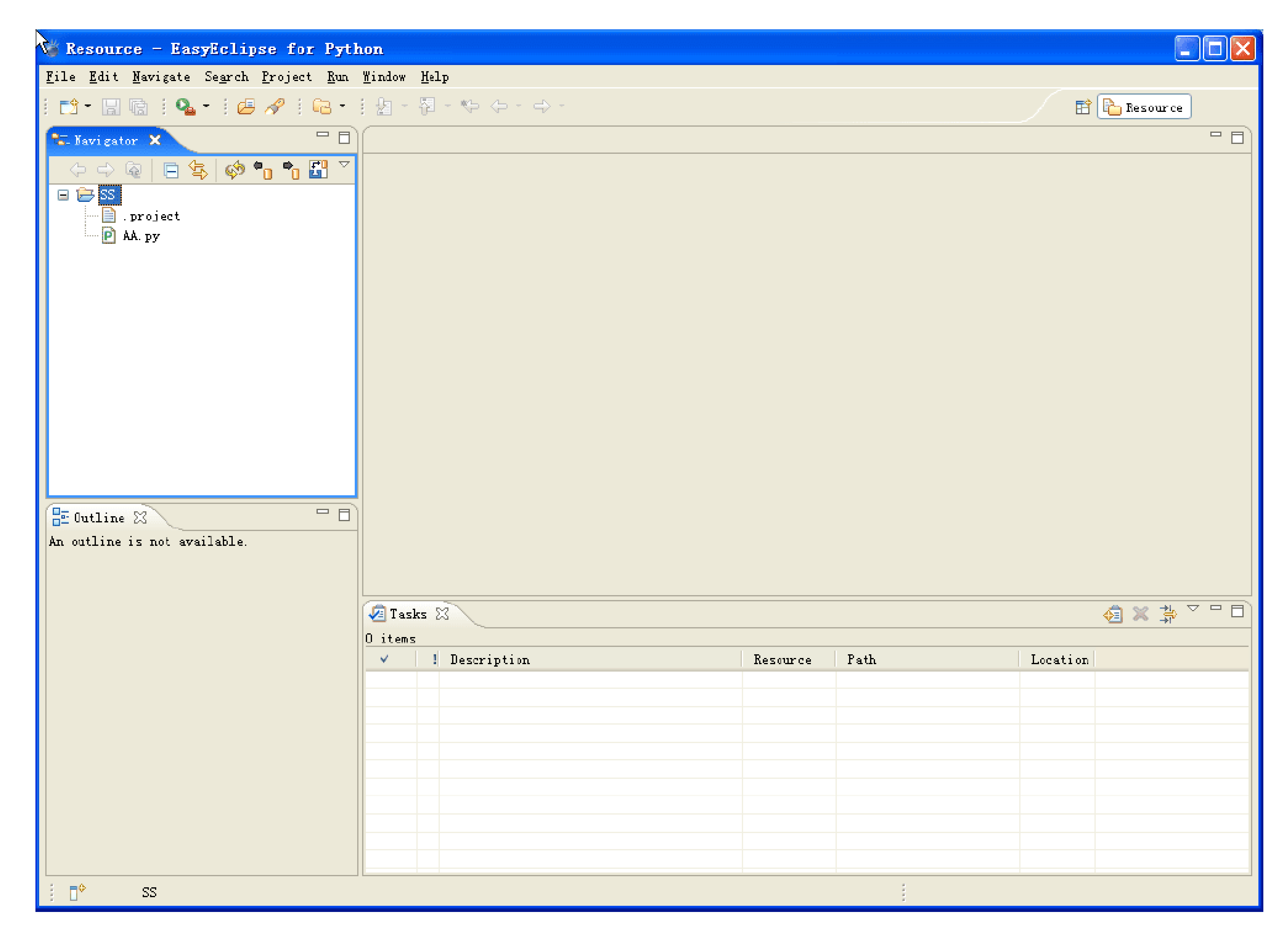


图 1-21 Eclipse IDE集成开发环境

新建一个名称为 Python 的群组，分别在“菜单文字”文本框中输入 Python，在“命令”文本框中输入 Python 的安装路径，在“参数”文本框中输入$(FileName)，在“初始目录”文本框中输入$(FileDir)，如图 1-22 所示。一定要在“动作”下拉列表框中选择“捕获输出”选项，只有这样 Python 程序运行后的输出结果才会显示在 EditPlus 的输出栏，否则运行 Python 程序后将弹出命令行窗口，并把结果输出到命令行中。单击“确定”按钮，新建一个 Python 文件，此时“工具”菜单下将会出现 Python 选项。单击 Python 选项或按快捷键 Ctrl+1，就可以运行 Python 程序了。

2) 增加高亮显示

为了使我们在编写代码时不至于对单色调的代码产生误解，可以对 EditPlus 编辑器增加高亮显示。但是在对编写的 Python 文件增加高亮显示之前，我们要用到两个文件：python.acp 和 python.stx(可以从 http://www.editplus.com/files/pythonfiles.zip 下载)。ACP 文件表示自动完成的特征文件，STX 文件表示语法加亮的特征文件。

当下载完成后，把下载的文件解压到 EditPlus 的安装目录下，然后在“参数选择”对话框中的“类别”选择框中选择“文件”|“设置 & 语法”选项。然后单击“添加”按钮，弹出“设置 & 语法”对话框，在该对话框中的文本框中输入 Python，单击“确定”按钮，Python 将出现在“参数选择”对话框中的“文件类型”选择框中。在“文件扩展名”文本框中输入 py，在“语法文件”文本框中输入 python.stx 的路径，在“自动完成”文本框中输入 python.acp 的路径，如图 1-23 所示，然后单击“确定”按钮，这样就完成了增加高亮显示的设置。

再次打开编辑器，就可以看到图 1-24 所示的效果。



图 1-22　添加Python工具



图 1-23　增加高亮显示

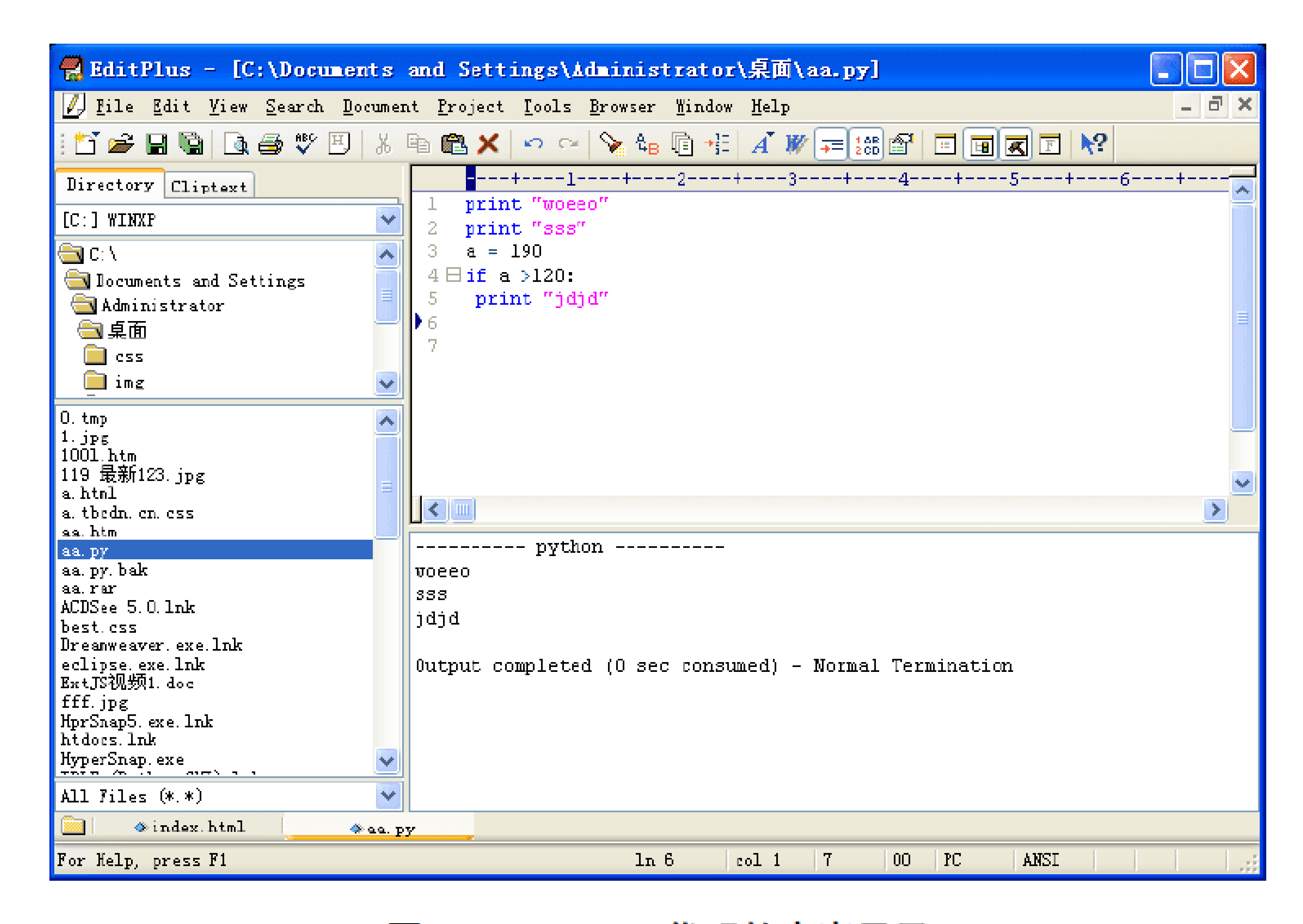
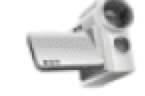


图 1-24　Python代码的高亮显示

1.5　保存并执行程序

在进行 Python 代码编程时，所用到的交互式解释器在退出时所有的编码都会消失。我们不仅要学会如何新建并编辑程序，还要学会保存文件。下面将讲解如何正确地保存和执行程序。

 **视频教学：光盘/videos/01/程序的保存和运行.avi**　　长度：6 分钟

1.5.1　基础知识——程序的保存和运行

在编写程序时，首先选择编辑器，这里我们使用 IDLE。打开 IDLE 后，可以通过执行 File | New Window 命令新建一个编辑窗口。在这个窗口里，输入以下代码：

```
name = raw_input("你叫什么名字：")
print "嗨，我的名字叫"+name
```

执行 File | Save 命令保存我们刚刚编写的程序。保存文件的路径一定要清楚，以及保存的文件必须以.py 结尾。

程序保存完毕，我们就可以运行它了。使用快捷键 Ctrl+F5，运行程序，将看到程序的运行结果，如图 1-25 所示。

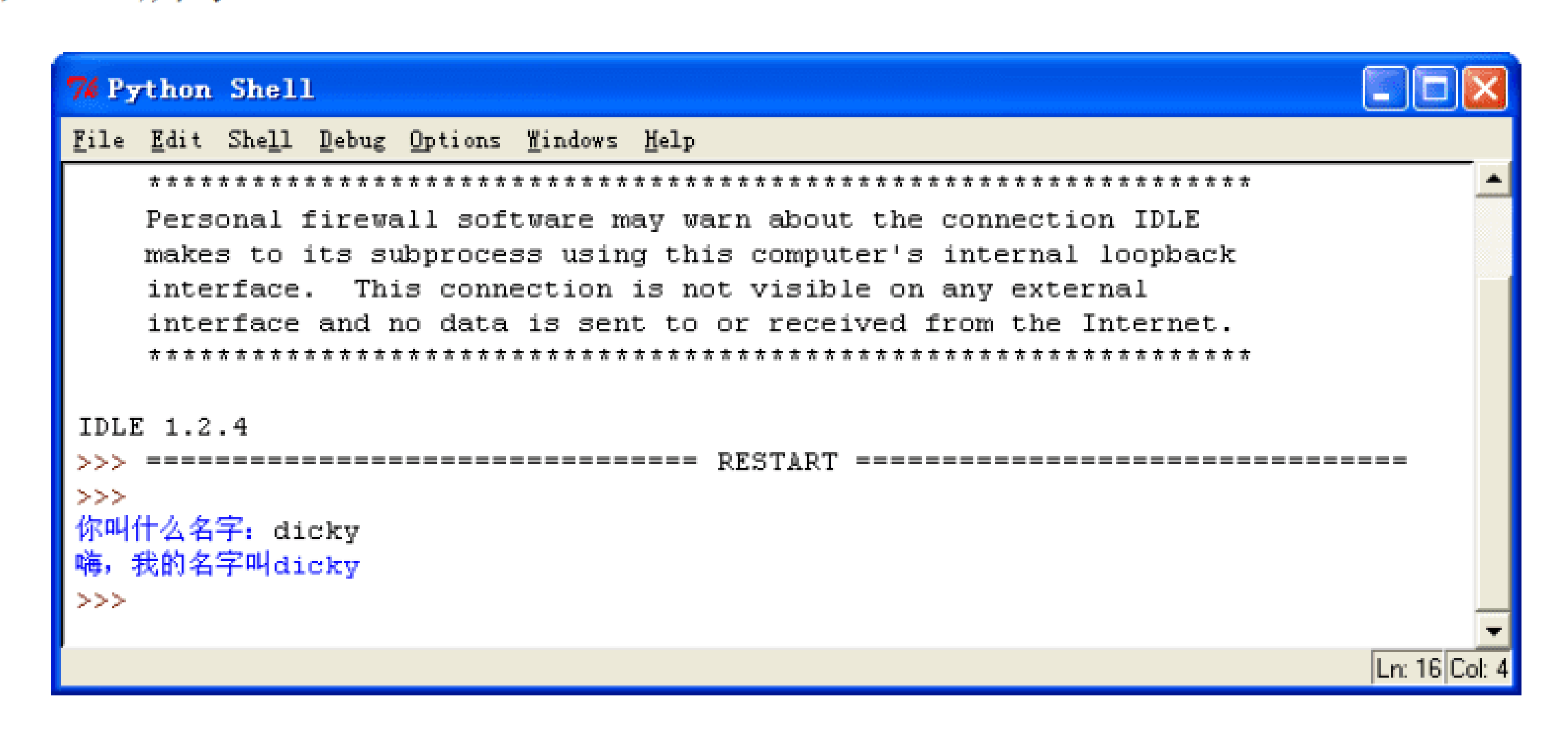


图 1-25　程序的运行结果

1.5.2　实例描述

我们虽然已经学习了 Python 的不少知识，但是还没接触到具体的 Python 程序如何编写。下面将编写 Python 的第一个程序，这个程序实现的功能是：对你输入的数字进行判断，到底是大于零还是小于零。

1.5.3　实例应用

【例 1-2】Python 的第一个程序。

(1) 新建一个名为 first.py 的文件，然后在文件中输入以下代码：

```
x = int(raw_input("Please enter an integer: "))
if x < 0:
    print 'this number < 0'
if x > 0:
    print "this number >0"
```

(2) 保存文件。

1.5.4　运行结果

使用快捷键 Ctrl+F5 通过 Python Shell 来执行文件，我们可以看到图 1-26 所示的结果。

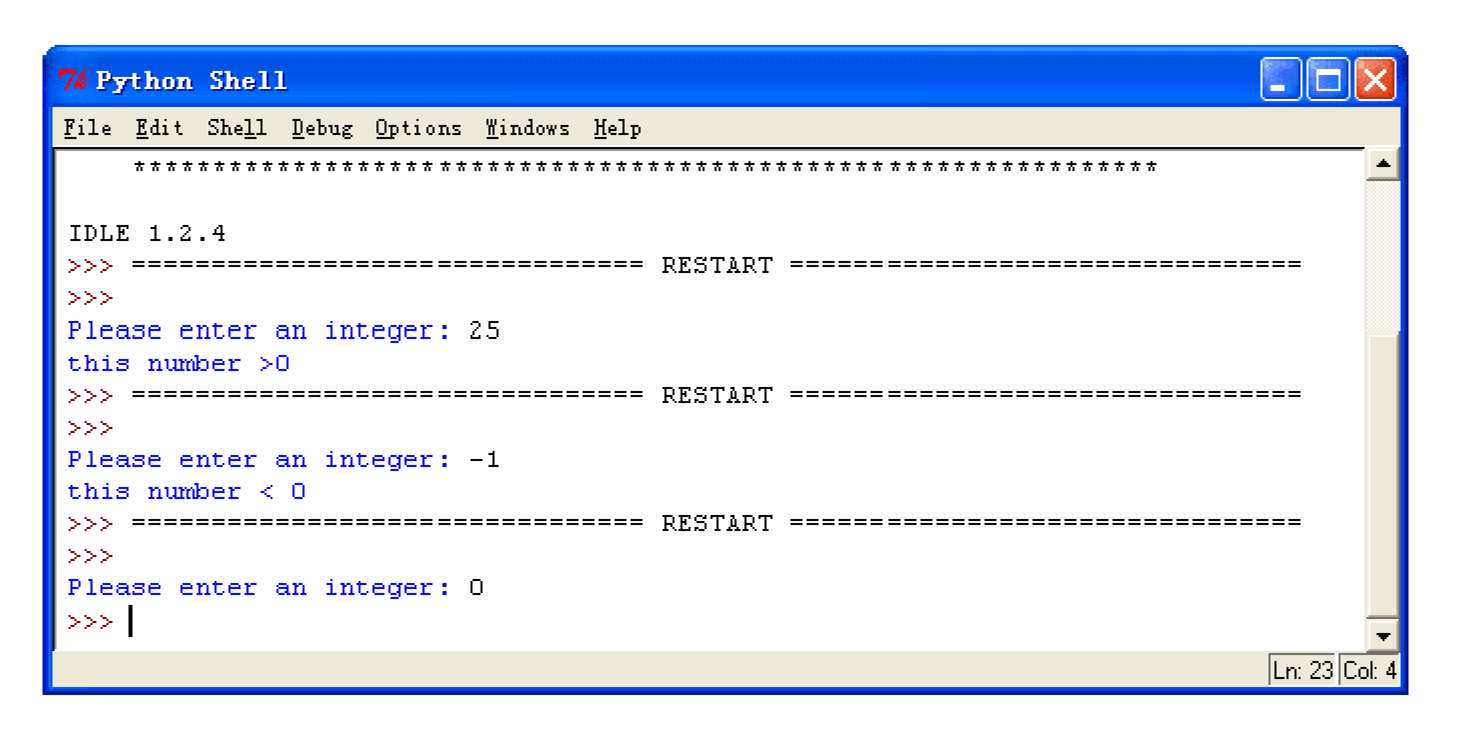


图 1-26　执行结果

1.5.5　实例分析



源码解析

在上述实例中我们通过使用 raw_input()函数来获取用户输入的信息，然后传给变量 x，通过对变量 x 的值进行判断，并给出执行响应的结果。

1.6　常见问题解答

1.6.1　关于Python版本的问题



关于 Python 版本的问题?
网络课堂：http://bbs.itzcn.com/thread-13631-1-1.html

我最近准备学 Python，发现 Python 2 和 Python 3 不兼容，那么先学 Python 2 好呢？还是直接学 Python3 好？

【解决办法】目前大多数应用都是 Python 2.5/2.6 居多。很多插件和库都还停留在 2.5/2.6 阶段，如果你去玩 3.0，会发现很多扩展都没有。所以说你现在玩 3.0 不实用。

Windows 上建议学 2.6，Linux 上就玩 2.5。Python 2 和 Python 3 并不是 100%不兼容，Python 3 只不过是对 Python 2 进行了一次大清洗，源码上精炼了许多，语法上做了一些修缮。会 Python 2 的人以后不会对 Python 3 感冒的。

1.6.2　Python的print问题



Python 的 print 问题?
网络课堂：http://bbs.itzcn.com/thread-13632-1-1.html

书上明明写的是 print 'hello world'，可是编译通不过，必须写成 print('hello world')才行，是怎么回事？是不是版本的问题？另外请说明一下 print 的常见用法。还有 raw_input 怎么用，写成 a = raw_input("please input a")怎么会报错，说是 raw_input 未声明，请解答。

【解决办法】第一个问题是版本的问题，你用的应该是 Python 3 吧，而 Python 3 刚刚出来，书上写的大多是 Python 2，所以会报错。你可以用 Python 2 或者以下版本试试。print 的常见用法就看书上写的好了。

第二个问题 raw_input 是将你的输入赋值给变量，例如：

```
a = raw_input("please input a")
```

在交互窗口运行时可以输出 please input a。在后边输入字符并回车，这样输入的字符就赋值给 a 了。

1.6.3　关于Python编程的问题



关于 Python 编程的问题?
网络课堂：http://bbs.itzcn.com/thread-13632-1-1.html

我读了一本 Python 的书，看完了，可这本书没有讨论语言的细节。编程理论也少得可怜，我想知道如果把这个语言学到中等水平应该读哪些书，有没有视频教程，希望指导一下学习的规划。

【解决办法】语言的细节基本上你看 Python 自带的 python documentation 就足够了，不过好像比较多，先看其中的 python tutorial 就可以了，然后再看 dive into python。如果要做一个好的程序员，算法是少不了的。但是这和 Python 无关。你可以把其他语言课中遇到的习题用 Python 实现，对初学者来说这是很好的锻炼方式。

1.7　习　　题

一、填空题

(1) Python 是在一个名为__________人手中诞生的。
(2) Python 成为继 C++、Java 之后的第__________种编程语言。
(3) Python 是一种__________的语言，并不需要编译，而是直接在机器上执行。

二、选择题

(1) 以下__________不是 Python 语言的特点。
A. 免费开源　　B. 运行于服务器端
C. 可移植性　　D. 解释性

(2) 在 Python 的应用领域，__________含有庞大的诸如 PIL、Tkinter 等图形类库的支持，能够方便地进行图形处理。
A. 系统编程　　B. 数字处理　　C. 文本处理　　D. 图形处理

(3) ___________的发行版本包括 Windows 应用程序接口和 COM 组件模型，它是世界上最早出现的 Python 开发工具之一。

A. PythonWin　B. Eclipse　C. IDLE　D. EditPlus

三、上机练习

上机练习： 自己动手安装和配置 Python，并安装一种开发工具，如 PythonWin。

练习要求：能够手动安装和配置 Python，并通过安装 PythonWin 达到能够编写代码的程度。



第 2 章　练就扎实的基本功

内容摘要

PHP 靠一个“简单”占领了市场，PHP 的哲学是快速而“不择手段”。Python 同样简单，但 Python 的哲学是快速而漂亮，它的漂亮体现在代码上。Python 是一种面向对象、直译式计算机程序设计语言，也是一种功能强大而完善的通用型语言，已经具有十多年的发展历史，成熟且稳定。这种语言具有非常简捷而清晰的语法特点，适合完成各种高层任务，几乎可以在所有的操作系统中运行。

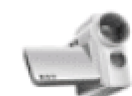
每一种语言都会有它独特的一些基本常识，比如变量、字符串的声明和使用，该语言的数据类型都有哪些？当用户需要向系统中输入一段字符时，该语言如何将用户所输入的字符输出？如何为代码添加注释。本章将介绍 Python 的一些基本常识。

学习目标

- 掌握变量的声明和使用。
- 掌握 Python 命令。
- 掌握 Python 的数据类型和表达式。
- 掌握向代码中添加注释。
- 了解 Python 的运算符。

2.1 Python的编码规则

每一种语言都有自己的编码规则，编程时不可违背这些准则。一旦违背，程序就无法执行，甚至出现异常。本节将讲解 Python 的一些编码规则。


视频教学：光盘/videos/02/ Python 的编码规则.avi　　长度：16 分钟

2.1.1 基础知识——代码缩进与冒号

代码缩进是指通过在一行代码的前面输入若干空格或者制表符来表示行与行之间的层次关系。每一种编程语言一般都需要代码缩进来规范程序代码的层次结构，使代码清晰，易于阅读和理解。对于多种语言，例如 C、C++、Java、C#等，代码缩进作为一种良好的编码习惯而延续下来。对 Python 语言来讲，代码缩进是一种语法，Python 语言中没有采用花括号或 begin...end 来分隔代码块，而是使用冒号和代码缩进来区分代码之间的层次。

使用 Eclipse IDE 开发工具或者 EditPlus 等编辑器书写代码时，编辑器会自动缩进代码，并在需要添加冒号的地方自动补充冒号，提高了编码效率，为程序员减轻了很多编写代码的负担。下面使用 EditPlus 编辑器来编辑一段代码，并采用代码缩进的语法来显示条件语句。

```
time=12
if(time==12):
    print '12'       #代码缩进
else:
    print '18'       #代码缩进
```

在上面的代码中，首先创建了变量 time，并赋值为 12。在这条语句中，赋值运算符=两侧各添加了一个空格，这是一种良好的编程习惯，提高了程序的可读性。接着使用 if 条件语句判断 time 的值是否为 12，在 if 条件语句之后输入一个冒号，而冒号后面的代码块则需要缩进编写，因为当 if 条件成立时，程序才能执行 if 块中的代码，因此第 3 行代码位于第 2 行代码的下一个层次。当启用 EditPlus 编辑器的自动缩进功能时，代码块 print '12'会由 EditPlus 自动缩进。下面的 else 语句是一段新的代码块，与 if 条件语句是同层结构，因此直接从最左端书写代码即可。运行上段代码，输出结果如下：

```
12
```

Python 对代码缩进要求很严格，如果程序中没有采用代码缩进的编码风格，程序将会抛出一个 IndentationError 的异常信息。



如果缩进的代码前只有一个空格或者几个制表符也是符合语法要求的，但是不推荐使用这种写法。最佳的方法就是编码前统一代码的书写规则，使所有代码前的空格数保持一致，最好使用 4 个空格缩进。

每行代码缩进的情况不一样，代码执行的结果也会有所不同，例如下面的代码：

```
time=12
if(time==12):
    print '12'              #代码缩进
else:
    print '18'              #代码缩进
    time=time+6             #代码缩进
    print str(time)         #代码缩进
```

执行该段代码，输出结果如下：

```
12
```

修改上面代码的缩进情况，代码如下：

```
time=12
if(time==12):
    print '12'              #代码缩进
else:
    print '18'              #代码缩进
time=time+6                 #代码缩进
print str(time)             #代码缩进
```

执行该段代码，输出结果如下：

```
12
18
```

从上面的两段代码可以看出，不同的代码缩进执行的结果不同。因此，当程序出现问题时，首先需要检查代码的书写是否正确，如果正确再检测代码缩进是否合理。

2.1.2　基础知识——使用空行分隔代码

函数与函数之间或者类与类之间用空行分隔，表示一段新的代码的开始。类和函数入口之间也用一行空行分隔，以突出函数入口的开始。

下面创建一个类 MyClass，并在类 MyClass 中定义 myFirstFun()和 mySecondFun()方法，代码如下：

```
class MyClass:
    def MyFirstFun (self):
        print 'MyFirstFun()'

    def MySecondFun (self):
        print 'MySecondFun()'

if __name__ == "__main__":
    myclass = MyClass()
    myclass.MyFirstFun()
    myclass.MySecondFun()
```

在上面的代码中，类 MyClass 中的第一个方法 MyFirstFun()与第二个方法 MySecondFun()之间插入了一个空行，在第二个方法结束与主程序的入口之间又插入了一个空行，便于阅读代码，区分功能结构。

用两行空行分隔顶层函数和类的定义，类内方法的定义用两个空行分隔，额外的空行可用

于分隔相关函数组成的群，在一组相关的单句中间可以省略空行。当空行用于分隔类中方法的定义时，在 class 行和第一个方法定义之间也要有一个空行。在函数中使用空行，表示一个逻辑段落。



Python 中的空行与代码缩进是完全不同的概念，空行并不是 Python 语法的一部分，只是程序代码的一部分。书写时不插入空行，Python 解释器运行是正常的，但是空行的作用在于分隔两段不同功能或者不同含义的代码，以便于程序的后期维护或重构。

2.1.3 基础知识——命名规则

每一种语言也会有一套属于自己的命名规则，Python 语言也不例外。下面介绍几种常见的命名规则。

1. 包、模块的命名规则

Python 语言中的包名与 Java 语言中包的命名规则是相同的，全部以小写字母的形式来命名。模块名应该是不含下划线、简短、小写字母这些规则来命名的，因为模块名被映射到文件名，有些文件系统对大小写不敏感并且会截取比较长的名字。例如：

```
Filename:pythonModule.py
```

该语句声明了一个模块的名称，模块名全部用小写字母组成。

2. 类、对象的命名规则

Python 中的类名采用 CapWords 约定，即每个单词的首字母要大写，其他字母小写，例如 MyStudent。对象名用小写字母来表示。类的私有属性、私有方法以两个下划线作为前缀，对象通过点操作符来访问类中的属性和方法。例如下面的代码：

```
class MyClass:       #类名中的每个单词的首字母大写，其他小写
    __username=''    #私有属性前必须使用两个下划线为前缀
    def __init__ (self,username):
        self.__username=username        #self 相当于 Java 语言中的 this 关键字，表示本类
    def getUserName (self):             #方法名的首字母小写，其后每个单词的首字母要大写
        return self.__username
if __name__ == "__main__":
    myclass = MyClass('admin')          #对象名用小写字母
    print myclass.getUserName()
```

在该段代码中，首先创建了 MyClass 类，类名的每个单词的首字母大写，其他小写。在类中定义了一个私有属性，以两个下划线为前缀命名，即__username。在类中定义了一个私有方法，同样采用两个下划线命名，即__init__。在方法中，使用了 self 为前缀来说明__username 属性属于 MyClass 类。在 MyClass 类中还定义了一个公有的方法，方法名的首字母小写，其后的每个单词首字母大写，其他字母小写。在该方法中，将类中的__username 属性使用 return 返回。在程序的入口函数中，创建了一个名称为 myclass 的对象，对象名小写。

3. 函数的命名规则

函数名的命名规则如下。

- 函数名必须以下划线或字母开头，可以包含任意字母、数字或下划线的组合。
- 函数名是区分大小写的。
- 函数名不能是保留字。

下面通过一段代码来详细了解函数名的命名规则。

```
import random
def equalseNum (num):    #函数名首字母小写，后面每个单词的首字母大写
    if(num == 6):
        print 1
    else:
        print 0
num=random.randrange(1,9)
print 'num = '+str(num)
print equalseNum(num)
```

在该段代码中，首先导入 random 模块，接着定义了一个函数 equalseNum()。该函数的名称首字母小写，后面每个单词的首字母大写。参数 num 接收的是下面自动生成的数字。在该函数中，使用 if 条件语句来判断参数 num 是否为 6，如果为 6，输出 1，否则输出 0；然后定义了一个变量来接收生成随机数的模块 random 中 randrange()函数所生成的 1～9 的数字，函数 randrange()以模块 random 作为前缀命名；最后将生成的数字作为参数传入到函数 equalseNum()中。

2.1.4　基础知识——为代码添加注释

注释是用于说明代码实现的功能，采用的算法，代码的编写者以及代码创建和修改的时间等信息。注释是代码的一部分，起到对代码补充说明的作用，易于程序的阅读分析。C、C++、Java 等语言均采用//或/*…*/作为注释的标记，Python 的注释方式有所不同。

1. 单行注释

Python 中的单行注释使用#号加若干空格开始，后面是注释的内容，以回车作为注释的结束。例如：

```
#声明并初始化变量 num
num=1
```

2. 行内注释

Python 中的行内注释是最常用的，行内注释应该至少用两个空格和语句分开，它们以#号和单个空格开始。例如：

```
num=1   #声明并初始化变量 num
```

3. 注释块

注释块通常应用于跟随一些(或者全部)代码并和这些代码有着相同的缩进层次。注释块中

也使用#号和一个空格开始。注释块内的段落以仅含单个#的行分隔。例如：

```
# 声明并初始化变量 num
# 改变变量 num 的值，使值扩大 10 倍
#
# 输出变量的值
num = 12
num *= 10
print num
```

Python 一般会忽略#行的内容，跳过#行执行后面的内容。特殊含义的注释例外。Python 还有一些特殊的注释，用以完成一些特别的功能，例如中文注释、程序的跨平台等。

1) 中文注释

如果需要在代码中使用中文注释，必须在 Python 文件的最前面加上如下注释说明：

```
#_*_ coding:UTF-8 _*_
```

2) 跨平台注释

如果需要使 Python 程序运行在 Windows 以外的平台上，则需要在 Python 文件的最前面加上如下注释说明：

```
!# /usr/bin/python
```

2.1.5 基础知识——语句的分隔

在 C、Java 等语言的语法中规定，必须以分号作为语句结束的标识。Python 也支持分号，同样用于一条语句的结束。但在 Python 中分号的作用已经不像 C、Java 中那么重要了，Python 中的分号可以省略，主要通过换行来识别语句的结束。例如：

```
print "my name is MaXiangLin"
print "my name is MaXiangLin";
```

这两行代码是等价的，输出的结果也是相同的。如果需要在一行代码中书写多条语句，就必须使用分号分隔每条语句，否则 Python 无法识别语句之间的间隔。例如：

```
# 使用分号分隔语句
x = 1 ; y = 2 ; z = 3
```

在该语句中有 3 条赋值语句，语句之间需要用分号隔开，如果不隔开，Python 解释器将不能正确解释，并提示如下语法错误：

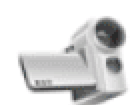
```
SyntaxError: invalid syntax
```



分号不是 Python 推荐使用的符号，Python 倾向于使用换行符作为每条语句的分隔，简单直白是 Python 语法的特点。通常一行只写一条语句，这样便于阅读和理解程序。一行写多条语句是不赞成使用的编码规范。

2.2 数　　值

交互式 Python 解释器可以当做功能非常强大的计算器使用，当需要计算两个数字类型的数据相加、相减等时，需要在 Python 解释器中输入两个数字，并使用运算符将其连接，形成一个表达式，最后得到计算的结果。本节将介绍 Python 中的数字类型以及如何灵活运用不同类型的数字计算出不同类型的值。

 **视频教学：**光盘/videos/02/数值.avi　　　　长度：10 分钟

基础知识——数值类型

数字提供了标量贮存和直接访问。它就是不可更改的一种数据类型，也就是说变更数字的值会生成新的对象。Python 支持 6 种数字类型，分别是整型、长整型、布尔型、双精度浮点型、十进制浮点型和复数。

1. 整型

Python 有 3 种整数类型，分别是布尔型、长整型和标准整数类型。其中，布尔类型只有两个值，即 1 表示 True，0 表示 False；常规的整型是绝大多数系统都能识别的整型；Python 也有长整数类型，不要将 Python 的长整数与 C 语言的长整数混淆。Python 的长整数所能表达的范围远远超过 C 语言的长整数。事实上，Python 长整数仅受限于用户计算机的虚拟内存的大小。换句话说，Python 能轻松表达无穷大的整数。下面来了解一下 Python 的整数类型的运算。

1) 布尔型

从 Python 2.3 开始，布尔类型被添加到 Python 的数据类型中。尽管布尔值看上去是 True 和 False，但事实上是整数类型的子类，不能再被继承而生成它的子类。尽管布尔值由常量 True 和 False 表示，如果将布尔值放到一个数值上下文环境中(比方将 True 与一个数字相加)，True 会被当做整数值 1，而 False 则会被当成整数值 0。复数(包括-1 的平方根，即所谓的虚数)在其他语言中通常不被直接支持(一般通过类来实现)。

2) 标准整数类型

Python 的标准整数类型是最通用的数字类型，在大多数 32 位机器上，标准整数类型的取值范围是-2 147 483 648 至 2 147 483 647。如果在 64 位机器上使用 64 位编译器编译 Python，那么在这个系统上的整数将是 64 位。在 Python 解释器中输入 sys.maxint 表示最大整数，-maxint-1 表示最小整数。Python 标准整数类型等价于 C 语言中的长整型。整数一般以十进制表示，但 Python 也支持八进制或十六进制。八进制整数以数字 0 开始，十六进制整数则以 0x 或 0X 开始。

3) 长整型

长整数类型是标准整数类型的超集，当程序需要使用比标准整数类型更大的整数时，长整数类型就有用武之地了。在一个整数值后面加个 L(大小写都可以)，就表示这个整数是长整数类型，它可以是十进制、八进制或十六进制。


提示

Python 的长整型类似于 Java 中的 BigInteger 类型。从长远来看，整型与长整型正在逐步统一为一种整数类型。从 Python 2.3 开始，就不会出现整型溢出的错误，结果会被自动转换为长整数。在未来版本的 Python 中，两种整数类型将会无缝结合，长整数后缀 L 也会变得可有可无。

Python 可以处理一些数值非常大的整数。

```
>>>8800000000000000000
8800000000000000000L
```

在上面的代码中，数字后面加了一个 L。普通整数不能大于 2 147 483 647(也不能小于−2 147 483 648)，诸如这么大的数，建议大家使用长整型来表示。长整型书写方法和普通整数一样，只是在结尾多了一个 L(这里也可以使用小写的“l”，只是小写的“l”容易与数字“1”混淆，因此不建议大家使用小写)。

在上面的代码中，Python 把整数转换为了长整型数，那么能不能对更庞大的数字进行计算呢？下面来看段代码：

```
>>> 123456789987654321L*123456789987654321L+123456789987654321
15241578994055784324188385777625362L
```

如上面代码所示，长整型数与标准整型数是可以混淆使用的。在绝大多数情况下，不用在意长整型数和标准整型数的区别，除非需要进行类型检查。


注意

如果使用 Python 2.2 以下的版本，会看到如下的代码片段：

```
>>>8800000000000000000
OverflowError:integer literal too large
```

2. 双精度浮点型

Python 中的浮点数类似于 C 语言中的 double 类型，是双精度浮点数，可以直接用十进制或科学计算法表示。每个浮点数占 8 个字节(64 比特)，完全遵守 IEEE 754 规范(52M/11E/1S)，其中 52 比特用于表示底，11 比特用于表示指数(可表示的范围大约是正负 10 的 308.25 次方)，剩下的 1 比特表示符号。实际精度依赖于机器架构和创建 Python 解释器的编译器。

浮点数的表示方法通常使用一个小数点和一个可选的后缀 E(大小写都可以，表示科学计数法)，在 E 和指数之间可以用正(+)或负(−)表示指数的正负(如果是正数，可以省略前面的+号)，例如：

```
-1.609E-19
```

3. 复数

一个实数(整数或负数)和一个虚数(虚拟的数)的组合构成一个复数，一个复数是一对有序浮点数(x,y)，表示为 x+yj，其中 x 是实数部分，y 是虚数部分。虚数是不能单独存在的，它总是和一个值为 0.0 的实数部分一起构成一个复数，并且实数部分和虚数部分都是浮点数，虚数部分必须有后缀 j 或 J。下面是一些复数的例子：

```
64.254+2.5j  0+2j  9.54847754-8.31441J
```

复数对象拥有的属性如表 2-1 所示。

表 2-1　复数属性

属　性	描　述
num.real	该复数的实数部分
num.imag	该复数的虚数部分
num.conjugate()	返回该复数的共轭复数对象

下面编辑一段代码，具体了解使用复数的不同属性得到的值是怎样的。在 Python 的解释器中编辑代码，如图 2-1 所示。

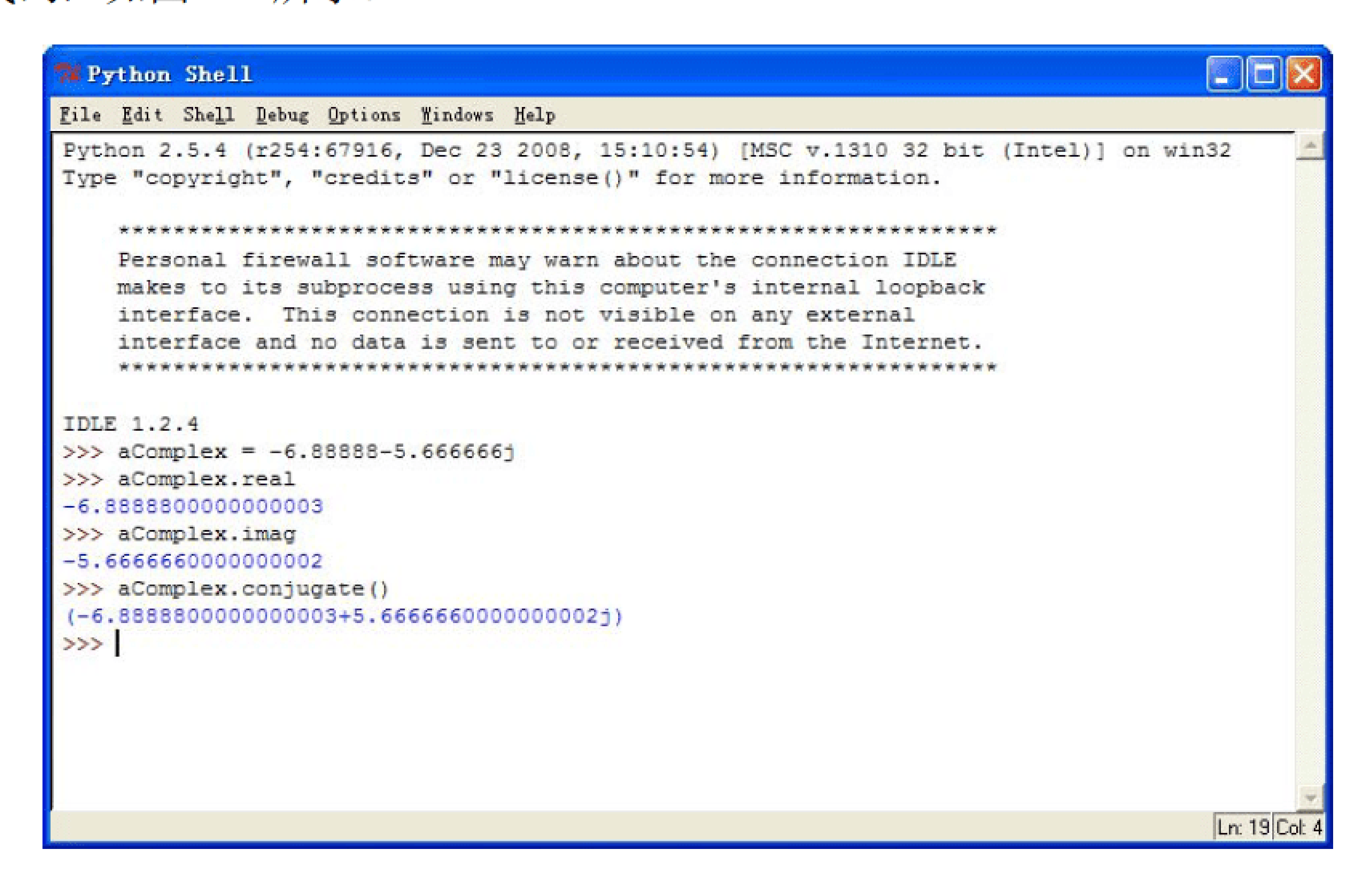


图 2-1　使用复数属性

4. 十进制浮点数

从 Python 2.4 起，十进制浮点数成为一个 Python 特性。十进制浮点型不是内建类型，必须先导入 decimal 模块才可以使用这种数值类型。例如，由于在二进制表示中有一个无限循环片段，在 Python 的解释器中输入 3.1，如下面的代码：

```
>>>3.1
3.1000000000000001
```

为什么会这样呢？因为绝大多数 C 语言的双精度实现都遵守 IEEE 754 规范，其中 52 位用于底。因此浮点值只能有 52 位精度，类似这样的值的二进制表示只能像上面那样被截断。在这种情况下就需要使用十进制浮点型来表示该数据。首先必须导入 decimal 模块以便使用 Decimal 类，代码如下：

```
>>>from decimal import Decimal
```

下面编辑一段代码，具体介绍 Python 中的十进制浮点数在应用中发挥着怎样的作用。代码片段如图 2-2 所示。

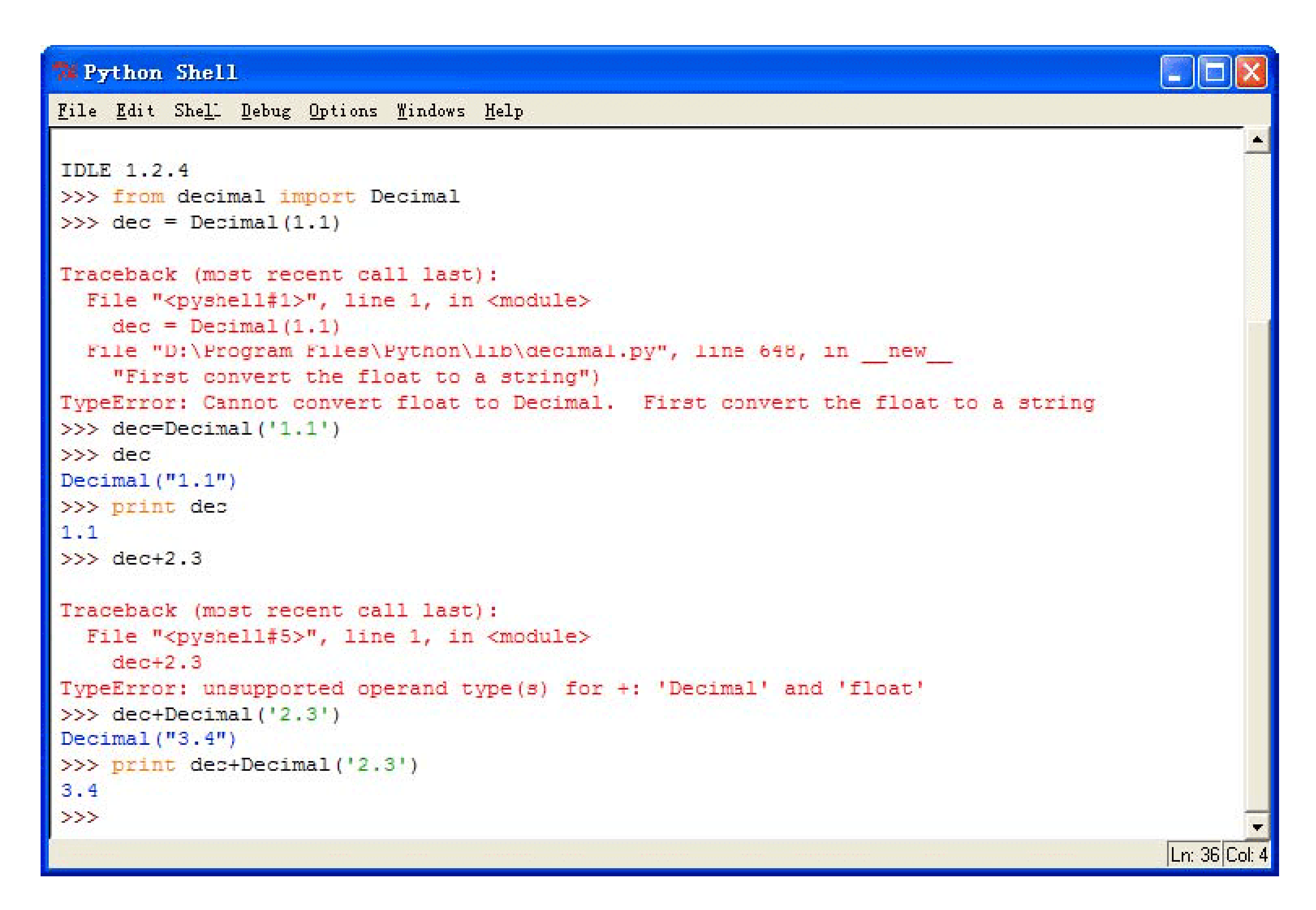
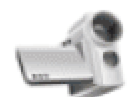


图 2-2　十进制浮点数的使用情况

2.3　制作超市购物清单

Python 中变量名称规则与其他大多数高级编程语言是相同的，都是受 C 语言的影响(可以说这门语言本身就是用 C 语言写成的)。本节将介绍 Python 中的变量与赋值。

 视频教学：光盘/videos/02/变量与赋值.avi　　　　长度：7 分钟

2.3.1　基础知识——标识符的命名

标识符是用来标识某种对象的名称。在命名标识符时，需要遵循下列规则。

(1) 标识符的第一个字符必须是字母(大小写均可)，或者是一个下划线(“_”)。

(2) 以下划线开头的标识符是有特殊意义的，其中：

- 以单下划线开头的(_foo)代表不能直接访问的类属性，需通过类提供的接口进行访问，也不能用 from xxx import * 导入。
- 以双下划线开头的(__foo)代表类的私有成员。
- 以双下划线开头和结尾的(__foo__)代表 Python 中特殊方法专用的标识，例如__init__()代表类的构造函数。

(3) 标识符名称的其他部分可以由字母(大小写均可)、下划线(“_”)或数字(0～9)组成。

(4) 标识符名称对大小写敏感。例如，count 和 Count 不是同一个标识符。

有效的标识符名称的例子，例如 i、j、_myname、my_name_123 或 abc23_d7。无效的标识符名称的例子，例如 this is spaced out 或 my-name。

2.3.2　基础知识——变量与赋值

变量是标识符的一个例子。变量名仅仅是一些字母开头的标识符，所谓字母开头，就是指以大小写字母开头，另外还可以是下划线开头，其他字符可以是数字、字母(大小写均可)或下划线。Python 中的变量名对大小写也是非常敏感的，也就是说 cOdE 与 CoDe 是两个不同的变量。

Python 是动态类型的语言，不需要预先声明变量的类型。变量基本上就是代表某值的名字。举例来说，如果希望用 i 代表 9，那么只需执行下面的语句即可：

```
>>>i=9
```

这样的操作称为赋值，即值 9 被赋给变量 i。变量的类型和值在赋值的那一刻就被初始化了。该语句表示：变量 i 的类型为 int 类型，初始化值为 9。变量赋值通过等号来执行，例如：

```
>>> price=99.9
>>> count=100
>>> name='Tom'
>>> total=price*count*0.8
```

上面是 4 个变量的赋值语句：第一个是浮点数赋值，第二个是整数赋值，第三个是字符串赋值，第四个是浮点数乘法赋值。Python 还支持递增或递减赋值，例如：

```
>>> count=count+1
>>> count
101
```

Python 同时也支持增量赋值，也就是将运算符和等号合并在一起，例如：

```
>>> count=count*10
>>> count
1010
```

可以改成增量赋值方式来表示，例如：

```
>>> count*=10
>>> count
1010
```



Python 不支持 C 语言中的自增 1(++)或自减 1(--)运算符，因为+和-也是单目运算符，Python 会将++n 解释为 n，将--n 解释为-(-n)，从而得到 n。

2.3.3　基础知识——局部变量

局部变量就是只能在函数或代码段内使用的变量。函数或代码段一旦结束，局部变量的生命周期也将结束，在函数或代码段外是调用不到的。局部变量的作用范围只在局部变量被创建的函数或代码段内有效。例如，在函数 myFun()中定义了一个局部变量，则该局部变量只能被 myFun()访问，其他函数或代码段无法访问 myFun()函数中定义的这个变量，如图 2-3 所示。

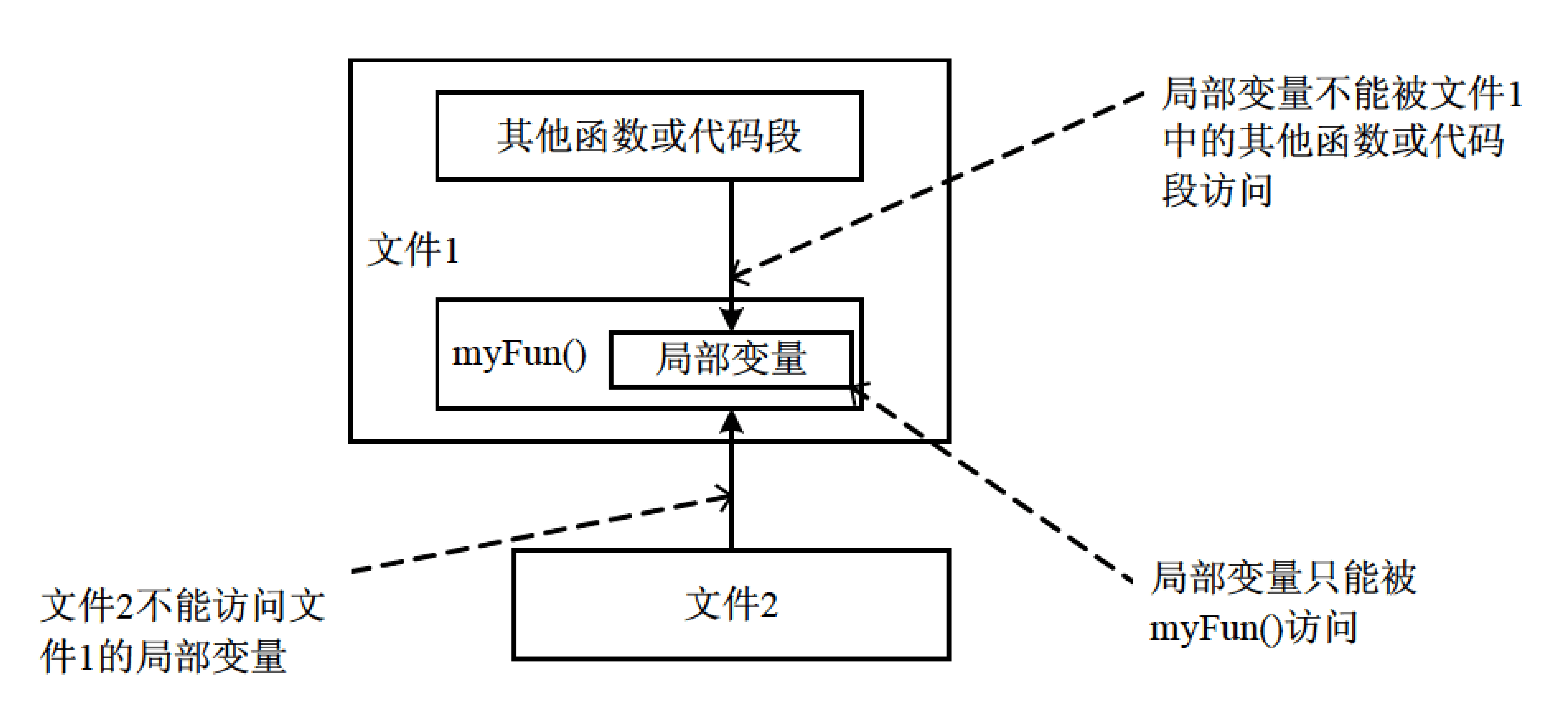


图 2-3　局部变量的作用范围

下面来看一个完整的例子。

```
class MyClass:          # 定义一个类 MyClass
    def myFun ():       # 在类 MyClass 中定义函数 myFun()
        num=12          # 在函数 myFun()中定义局部变量 num
        print 'myFun num='+str(num)
    def myFun2 ():      # 在类 MyClass 中定义另一个函数 myFun2()
        num=num+1       # 在函数 myFun2()中调用 myFun()函数中的局部变量 num，并重新赋值
        print 'myFun2 num='+str(num)
    num*=10             # 在类 MyClass 中调用 myFun()函数中的局部变量 num，并重新赋值
    print 'MyClass num='+str(num)
```

在该段代码中，首先定义了一个名称为 MyClass 的类，然后在类中定义了两个函数，分别为 myFun()和 myFun2()。在函数 myFun()中定义了局部变量 num，并在 myFun()函数中使用 print 语句将其调用并输出，输出的结果为：

```
myFun num=12
```

接着在 MyClass 类中定义了另一个函数 myFun2()，并在该函数中调用 myFun()函数中的局部变量 num，使其值发生改变，最后输出的结果为：

```
NameError: name 'myFun2' is not defined
```

最后在 MyClass 类中调用了 myFun()函数中的局部变量 num，并使值扩大 10 倍，最后输出的结果如下：

```
num*=10
NameError: name 'num' is not defined
```



Python 创建的变量就是一个对象，Python 会管理变量的生命周期。Python 对变量的回收也是采用垃圾回收机制。

2.3.4　基础知识——全局变量

全局变量是能够被不同的函数、类或文件调用的变量，在函数之外定义的变量即为全局变量。全局变量默认可以被文件内部的任何函数或任何代码段访问，外部文件也可以访问。但是，如果设置了该变量为私有变量，则外部文件是不可以调用的。例如，在文件 1 中定义了一个全局变量，文件 1 中的所有函数是可以访问该全局变量的，此外，对于文件 1 以外的文件也可以访问，如图 2-4 所示。

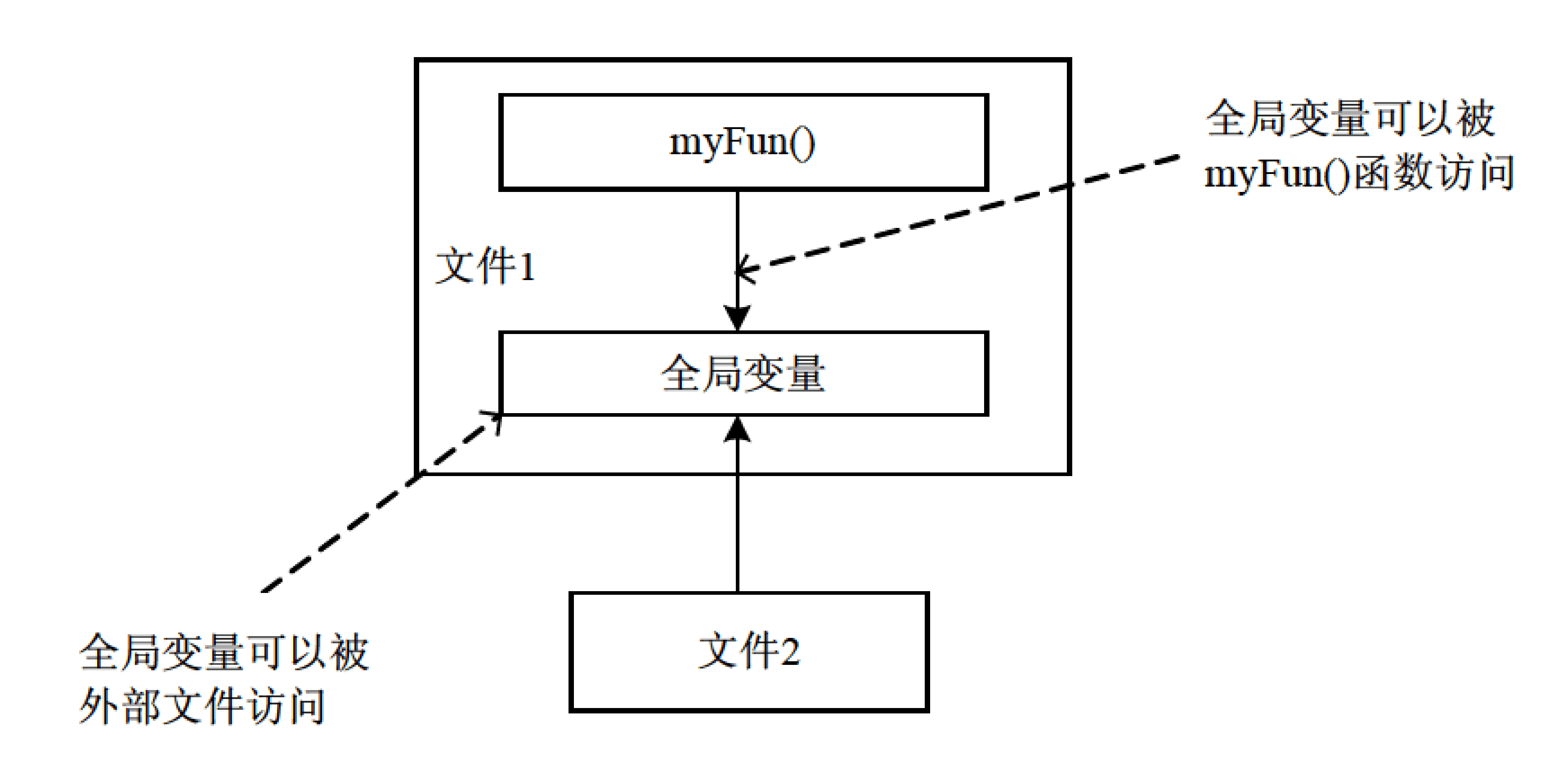


图 2-4　全局变量的作用范围



全局变量通常在文件的开始处定义。

下面再来看一段代码。

```
_num = 12                # 在文件的开头定义全局变量
def myFun ():            # 在文件中定义函数 myFun
    global _num          #  这里使用了 global 保留字，用于引用全局变量
    num = _num+1         # 调用全局变量，使值增 1 后赋给变量 num
    print 'myFun num='+str(num)
myFun()                  # 调用函数 myFun()
```

在该段代码中，首先在文件的开头定义了一个全局变量_num，并初始化值为 12。接着在文件中定义了名称为 myFun()的函数，在该函数中使用了 global 保留字，用于引用全局变量，使全局变量的值增 1 后赋值给 num 变量，并将 num 变量的值输出。在文件的最后调用了函数 myFun()。执行该段代码，输出的结果如下：

```
myFun num=13
```



如果不使用 global 保留字引用全局变量，有时候会出现不正常的结果。因此在引用全局变量时，要先使用 global 保留字将其引用。

2.3.5 实例描述

本人比较喜欢购物，基本上是每逢周末都会向超市投入一大笔资金，每次看着购物清单上的共计×××元，我都心疼得要命。下面给大家分享一下这周的购物清单吧。

2.3.6 实例应用

【例 2-1】 制作超市购物清单。

(1) 声明变量，以 pro_name_num 的形式命名标识商品名称的变量，用以记录商品名称。初始化商品 1 的名称，语句如下：

```
>>> pro_name_1='特仑苏'
```

(2) 声明变量，以 pro_price_num 的形式命名标识商品单价的变量，用以记录商品的单价。初始化商品 1 的单价，语句如下：

```
>>> pro_price_1=48
```

(3) 因为购买的“特仑苏”的数量为一箱，所以这里不再标识。声明计算“特仑苏”总价的变量，以 pro_total_num 的形式来标识，计算语句如下：

```
>>> pro_total_1=pro_price_1*1
```

(4) 按照同样的步骤，声明并初始化商品 2 的名称、单价，同时计算出商品 2 的总价，语句如下：

```
>>> pro_name_2='卡通垃圾桶'
>>> pro_price_2=2.5
>>> pro_total_2=pro_price_2*1
```

(5) 还是同样的步骤，声明并初始化商品 3 的名称和单价，不同的是，商品 3 的数量不是 1，需要同商品的名称和单价一样声明并初始化，购买的商品数量以 pro_count_num 的形式为变量来标识。商品 3 的购买清单如下：

```
>>> pro_name_3='舒洁面巾纸'
>>> pro_price_3=3.3
>>> pro_count_3=4
>>> pro_total_3=pro_price_3*pro_count_3
```

(6) 继续编写商品 4 和商品 5 的购买清单，语句如下：

```
>>> pro_name_4='桂格玉米味燕麦片'
>>> pro_price_4=23
>>> pro_total_4=pro_price_4*1
>>> pro_name_5='dove 洗面奶'
>>> pro_price_5=16.8
>>> pro_count_5=4
>>> pro_total_5=pro_price_5*pro_count_5
```

(7) 最后将 5 件商品的清单打印出来，语句如下：

```
>>> print '---------------购物清单---------------';\
     print '商品名称'+'  '+'数量'+'  '+'单价'+'  '+'总计';\
     print pro_name_1+'  '+'1'+'  '+str(pro_price_1)+'  '+str(pro_total_1);\
     print pro_name_2+'  '+'1'+'  '+str(pro_price_2)+'  '+str(pro_total_2);\
     print pro_name_3+'  '+str(pro_count_3)+'  '+str(pro_price_3)+'
'+str(pro_total_3);\
     print pro_name_4+'  '+'1'+'  '+str(pro_price_4)+'  '+str(pro_total_4);\
     print pro_name_5+'  '+str(pro_count_5)+'  '+str(pro_price_5)+'
'+str(pro_total_5)
```

2.3.7　运行结果

将上面的代码编辑之后，按回车键，Python 解释器将打印出购物清单列表，内容如下：

```
---------------购物清单---------------
商品名称  数量  单价  总计
特仑苏  1  48  48
卡通垃圾桶  1  2.5  2.5
舒洁面巾纸  4  3.3  13.2
桂格玉米味燕麦片  1  23  23
dove 洗面奶  4  16.8  67.2
```

2.3.8　实例分析



源码解析

在上面的例子中，声明的所有变量都是以字母开头。这里需要注意的是，变量只能以字母(大小写均可)或下划线("_")开头，而不能以数字开头，例如 9pro_name 是错误的。"\"表示换行符，如果需要换行，只需要在句尾添加\即可。

2.4　用户登录验证

Python 中的字符串被定义为引号之间的字符集合。几乎可以保证在每个 Python 程序中都需要用到字符串，可想而知字符串在一种语言中占据着很重要的位置。本节介绍 Python 中字符串的声明和使用。



视频教学：光盘/videos/02/字符串.avi　　　　长度：19 分钟

2.4.1 基础知识——字符串的声明与表示

Python 支持使用成对的单引号或双引号，而三引号(3 个连续的单引号或者双引号)可以用来包含特殊字符。那么，字符串与字符串的连接使用什么符号呢？字符串都有哪些比较常用的函数呢？下面将针对这些内容展开详细的讲解。

1. 单引号、双引号和三引号字符串

无论是单引号字符串、双引号字符串或者三引号的字符串，在 Python 程序中都是缺一不可的，它们各司其职。

1) 单引号字符串

使用单引号指示普通的字符串，例如：

```
>>> 'my name is MaXiangLin'
'my name is MaXiangLin'
```

所有的空白，即空格和制表符都照原有的样子保留，标识普通的字符串(不带任何引号的字符串)。

2) 双引号字符串

在双引号中的字符串与单引号中的字符串的使用是相同的，例如：

```
>>> "my name is MaXiangLin"
'my name is MaXiangLin'
```

这里让人吃惊的是：当 Python 打印出字符串时发现，使用双引号的字符串打印出来的却是使用了单引号的字符串，这有什么区别吗？事实上，这里无论是使用了单引号的字符串还是使用了双引号的字符串，打印出来的结果都是一样的，没有任何区别。但是在某些情况下，两者是必须同时使用的，缺一不可。

```
>>> "what's your name"
"what's your name"
>>> '"my name is MaXiangLin" I say'
'"my name is MaXiangLin" I say'
```

在上面的代码中，第一条语句使用了单引号的字符串(即撇号)，这时候就不能再使用单引号将整个字符串括起来了。如果使用单引号将其括起来，Python 的解释器是无法解释的，会输出如下的错误信息：

```
>>> 'what's your name'
SyntaxError: invalid syntax
```

第二条语句使用了单引号将整个字符串括了起来，其中包含了一个双引号的字符串。

3) 三引号字符串

使用三引号的字符串可以指示一个多行的字符串，并且在这个三引号的字符串中可以自由地使用单引号和双引号。例如：

```
>>> '''This is a multi-line string.This is the first line.
This is the second line.
"What's your name?" she said.
I say"my name is MaXiangLin"'''
```

```
'This is a multi-line string.This is the first line.
This is the second line.
"What\'s your name?" she said.
I say"my name is MaXiangLin"'
```

2. 转义引号字符串

假设需要在一个字符串中包含一个单引号，那么如何指示这个字符串呢？例如字符串 What's your name?。前面已经讲解过，可以使用双引号将字符串 What's your name? 括起来指示它，而不可以用'What's your name?'来指示，因为使用单引号将它括起来，Python 解释器将不明白这个字符串从何处开始，到何处结束。能指明这个字符串的开始位置和结束位置，除了使用双引号将其括起来之外，还可以通过转义符来完成这个任务，即使用'\'来指示单引号，例如：

```
>>> 'what\'s your name'
"what's your name"
```



对于在双引号字符串中使用双引号的情况，也可以借助转义符来表示。另外，可以使用转义符\\来指示反斜杠\本身。

需要注意的是，在一个字符串中，行尾出现一个反斜杠表示字符串换行，下一行继续，而不是开始一个新的行。例如：

```
>>> "This is the first.\
This is the second."
'This is the first.This is the second.'
```

3. 字符串的连接

下面先来看一段代码，即通过另外一种方式输出同样的字符串。

```
>>> 'what\'s your name' "my name is MaXiangLin"
"what's your namemy name is MaXiangLin"
```

在上面的代码中，只是编写了两个字符串，Python 会自动把这两个字符串连接在一起，合成一个字符串输出。不过，它只是在同时编写了多个字符串并且需要一个字符串紧接着另一个字符串的情况下才有效。例如，下列情况是无效的：

```
>>> a='what\'s your name?'
>>> b='my name is MaXiangLin'
>>> a b
SyntaxError: invalid syntax
```

通过上面代码可以发现，直接将多个字符串紧接着输出并不是连接字符串的方法，只是书写字符串的一种特殊方式而已。那么，如何连接字符串呢？在 2.3.6 节的案例中，我们已经接触过字符串的连接，那就是使用+号即可将多个字符串连接在一起。例如：

```
>>> 'what\'s your name?'+'my name is MaXiangLin'
"what's your name?my name is MaXiangLin"
>>> a='what\'s your name?'
>>> b='my name is MaXiangLin'
>>> a+b
"what's your name?my name is MaXiangLin"
```

4. 字符串的表示函数

Python 提供了两个函数来表示字符串。

- str()函数：把值转换为合理形式的字符串，以便用户理解。
- repr()函数：创建一个字符串，以合法的 Python 表达式的形式来表示值。

下面使用这两个函数来创建一些例子，具体了解一下它们的使用方法。

```
>>> print repr('my name is MaXiangLin')
'my name is MaXiangLin'
>>> print repr(123456L)
123456L
>>> print '-------------'
-------------
>>> print str('my name is MaXiangLin')
my name is MaXiangLin
>>> print str(123456L)
123456
```

在上面的代码中，首先使用了 repr()函数将字符串和长整型的数值以合法的 Python 表达式的形式表示，然后使用 print 语句输出，输出的结果保持了原有状态。接着使用 str()函数将字符串和长整型的数值转换为字符串，并使用 print 语句输出，输出的字符串不再带有单引号，长整型不再带有 L 字符，而是以字符串的形式输出。

简而言之，str()和 repr()函数是将 Python 的值转换为字符串的两种方式，其中函数 str()使字符串更易于阅读，而 repr()则把结果字符串转换为合法的 Python 表达式。

2.4.2 基础知识——输入与输出

字符串的输入与输出在 Python 程序中也是必不可少的，下面详细介绍 Python 中字符串的输入与输出。

1. 字符串的输出语句——print语句

细心的同学可能已经注意到，所有通过 Python 打印的字符串都是被引号括起来的，Python 打印时会保持用户输入字符串的原有状态，而不是单独将用户输入的字符串打印出来。那么，如何改变这种状况呢？下面使用 print 语句进行输出。

```
>>> 'my name is MaXiangLin'
'my name is MaXiangLin'
>>> 123456L
123456L
>>> print 'my name is MaXiangLin'
my name is MaXiangLin
>>> print 123456L
123456
```

在上面的代码中，先使用了单引号将字符串输出，输出的结果保持了原有的状态；接着输入一个长整型的数字，Python 解释器将输入的长整型包含字符 L 输出。接着使用 print 语句将与上面相同的字符串和长整型数字输出，但输出结果中不再包含单引号和字符 L，也就是说长整型数 123456L 被转换成了数字 123456，而且在显示时将转换后的数字 123456 输出。但使用

print 语句后，可能会对该数值是整型还是长整型产生不明确的概念。

2．字符串的输入函数——input()和raw_input()函数

有时候往往需要用户向程序中输入数据，这时就需要用到 Python 的 input()和 raw_input()函数。其中，input()函数是把读入的用户输入数据默认为 Python 表达式，而 raw_input()函数是把读入的数据转换为字符串。下面来看一段代码。

```
>>> input('what\'s your name?')
what's your name?'my name is MaXiangLin'
'my name is MaXiangLin'
>>> input('你的年龄：')
你的年龄：23
23
```

在上面的代码中，首先使用了 input()函数询问用户的名字？按回车键后，Python 会将用户的询问输出，同时等待用户回答，而当用户的回答是字符串时，需要使用引号(双引号、单引号或三引号)将回答的内容括起来。然而，要求用户使用引号输入内容是不合理的。下面再来看一段代码，了解一下使用 input()函数的不足之处。

```
>>> name= input('what\'s your name?')
what's your name?'MaXiangLin'
>>> print "my name is"+name+"!"
my name isMaXiangLin!
>>> age=input('你的年龄：')
你的年龄：23
>>> print "我今年"+age+"岁了！"

Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    print "我今年"+age+"岁了！"
TypeError: cannot concatenate 'str' and 'int' objects
```

在上面的代码中，首先使用名称为 name 的变量来存储用户输入的字符串 MaXiangLin，然后使用+连接符将 name 值与其他字符串连接并使用 print 语句输出，这样是正确无误的。再看下面的语句，使用了名称为 age 的变量来存储用户输入的年龄数值 23，然后也使用+号连接符将其连接，却出现错误，提示需要使用 str()函数将 int 类型的数值转换成字符串。很明显，使用 input()函数只是将读取的用户输入数据转换成 Python 合法的表达式。

诸如上面两种情况，体现了使用 input()函数的缺点，因此需要使用 Python 的另一个函数——raw_input()，它会把所有的输入当作原始数据，然后将其放入字符串中。

```
>>> raw_input('what\'s your name?')
what's your name?my name is MaXiangLin
'my name is MaXiangLin'
>>> raw_input('我的年龄：')
我的年龄：23
'23'
>>> age=raw_input('我的年龄：')
我的年龄：23
>>> print "我今年"+age+'岁了！'
我今年 23 岁了！
```

在接收用户输入数据时，一般情况下使用 raw_input()函数而不使用 input()函数。

2.4.3 实例描述

在 Web 应用程序中往往需要用户登录，才可以进入系统的主页面。实现用户登录就需要用户输入用户名和密码，然后检测用户输入的数据是否合法，只有检测通过才能让用户登录。下面就来模拟一下这个功能。

2.4.4 实例应用

【例 2-2】检测用户输入数据，实现用户登录功能。

(1) 声明变量 username，并使用 raw_input()函数接受用户输入的数据，语句如下：

```
>>> username=raw_input('您的用户名：')
```

(2) 按回车键，将字符串“您的用户名：”输出，并等待用户输入。

```
您的用户名：
```

(3) 用户在冒号后输入 admin 字符串。

```
您的用户名：admin
```

(4) 接着声明变量 password，并使用 raw_input()函数接受用户输入的数据，语句如下：

```
>>> password=raw_input('您的密码：')
```

(5) 按回车键，将字符串“您的密码：”输出，并等待用户的输入。用户在冒号后输入 admin 字符串。

```
您的密码：admin
```

(6) 使用 if 语句判断用户输入的用户名和密码是否合法，如果合法，输出用户登录成功信息。语句如下：

```
>>> if(username=='admin')and (password=="admin"):
    print "恭喜您！您输入的用户名和密码是合法的，登录成功！"
```

这里使用了 if 控制语句，在第 3 章中将具体讲解该控制语句的使用方法，这里暂不解释。

2.4.5 运行结果

编写完上面的 Python 语句后，当前用户名和密码均为 admin，符合登录用户的权限，即输出用户登录成功信息。按回车键，在 Python 解释器中输出下列结果：

```
恭喜您！您输入的用户名和密码是合法的，登录成功！
```

2.4.6 实例分析



源码解析

在上面的例子中，分别使用了 raw_input()函数来接受用户输入的用户名或密码，并使用了两个变量分别将用户输入的用户名和密码存储起来。接着使用 if 控制语句，检测用户输入的用户名或密码是否合法，在 if 语句块中使用了 print 语句输出用户登录成功的提示信息。

2.5　计算圆的周长和面积

Python 的运算符包括赋值运算符、算术运算符、关系运算符和逻辑运算符。表达式就是将不同类型的数据(包括常量、变量和函数)用运算符按一定的规则连接起来的式子。本节将详细介绍 Python 运算符与表达式的使用方法。

**视频教学**：光盘/videos/02/ Python 中的运算符与表达式.avi　长度：13 分钟

2.5.1　基础知识——算术运算符与算术表达式

算术运算符包括四则运算符、求模(余)运算符和求幂运算符。Python 中的算术运算符和表达式如表 2-2 所示。

表 2-2　Python中的算术运算符和表达式

算术运算符	算术表达式	描　述
+	x + y	加法运算
–	x – y	减法运算
*	x * y	乘法运算
/	x / y	除法运算
%	x % y	求模(余)运算
**	x**y	求幂运算

解释器的行为就像一个计算器，可以向它输入一个表达式，它会返回结果。表达式的语法简明易懂，+、–、*、/和大多数语言中的用法相同，用于计算，括号用于分组。例如：

```
>>> 5+5
10
>>> (200-50*2)/25
4
```

在上面的代码中，首先计算 5+5 的结果为 10，接着使用括号将其 200-50*2 先计算出结果再除以 25，结果为 4。在绝大多数情况下，常用的算术运算符的功能和计算器的功能是相同的。但 Python 存在一个潜在的陷阱——整数除法，下面看一段代码。

```
>>> 3/6
0
```

发生了什么呢？一个整数(无小数部分的数字)被另外一个整数(无小数部分的数字)整除，

计算结果的小数部分也被截除了，只留下整数部分。在 Python 2.5 版本中，当使用 x/y 形式进行运算时，如果 x 和 y 是整数，则对运算的结果进行截取，只留整数部分。下面再看一段代码。

```
>>> 3/-6
-1
```

顾名思义，3/-6 的商为-0.5，返回的是比-0.5 更小的且最接近的数值-1。

有些时候，这个功能还是比较可用的，但通常情况下只需要普通的除法即可。那么怎么实现呢？这里提供了两种有效的解决方案：用实数(包含小数点的十进制数)而不是整数进行运算，或者让 Python 改变除法的执行方式。实数在 Python 中被称为浮点数(float)，如果参与除法的两个数中有一个数为浮点数，结果亦为浮点数。

```
>>> 3.0/6.0
0.5
>>> 3.0/6
0.5
>>> 3/6.
0.5
```

从 Python 2.2 开始，除法运算符除了/之外，还引入了另一个除法运算符//(地板除)，后一种运算符只用于整除法。对于除法运算符/，默认时的行为跟 Python 2.2 之前一样，它视操作数而定，既可以进行整除，也可以进行真除法。如果想让这两个运算符有一个明确的分工，即/只用于真除法，而//仅用于整除法，则需要作以下声明：

```
from __future__ import division
```

下面来看一下两种除法运算符在作以上声明前后的区别。

```
>>> 5/6
0
>>> 5//6
0
>>> 5.0/6
0.83333333333333337
>>> 5.0//6
0.0
>>> from __future__ import division
>>> 5/6
0.83333333333333337
>>> 5//6
0
>>> 5.0/6
0.83333333333333337
>>> 5.0//6
0.0
```

在该段代码中，在声明之前，对表达式 5/6 进行计算时，结果为 0，这是因为参加运算的两个操作数都是整数，运算符/进行的是整除法。而表达式 5.0/6 的结果却是 0.83333333333333337，这是因为操作数中的 5.0 是浮点型数字，所以运算符/进行的是真除法。表达式 5//6 和 5.0//6 进行求值时，进行的都是整除法，不过返回值一个是整型，另一个是浮点型。当使用 import 语句之后，除法运算符/只能用于真除法，因此 5/6 和 5.0/6 的返回值都是真正的商。

到目前为止，我们已经了解了基本的算术运算符(加、减、乘和除)。除此之外，还有一个

非常有用的运算符——求余运算符%。先来看下面的代码。

```
>>> 3%6
3
>>> 100%30
10
>>> 100%20
0
>>> 2.75%0.5
0.25
```

在上面的代码中，3%6 得到的商为 0，余数为 3，因此输出结果为 3；100%30 得到的商为 3，余数为 10，因此输出结果为 10；而浮点数 2.75%0.5 得到的商为 5，余数为 0.25，因此输出结果为 0.25。

最后一个运算符就是幂(乘方)运算符**，先看一段代码。

```
>>> 3**3
27
>>> -3**2
-9
>>> (-3)**2
9
```



幂运算符比取反运算符(一元减运算符)的优先级要高，所以-3**2 与-(3**2)是等价的。

2.5.2　基础知识——关系运算符与关系表达式

关系运算符是对两个对象进行比较的符号。Python 中的关系运算符和表达式如表 2-3 所示。

表 2-3　Python中的关系运算符和表达式

关系运算符	关系表达式	描　述
<	x < y	小于
>	x > y	大于

续表

关系运算符	关系表达式	描　述
<=	x <= y	小于或等于
>=	x >= y	大于或等于
==	x == y	等于
!= 或 <>	x != y 或 x <>y	不等于

下面演示一下关系运算符的逻辑输出：

```
print 9 > 6
print 2 >= 2
print 9 == 9
print 2!=2
print 6<>9
```

在第 1 行代码中，9>6 的逻辑关系成立，输出结果为：

```
True
```

在第 2 行代码中，2>=2 的逻辑关系成立，输出结果为：

```
True
```

在第 3 行的代码中，9==9 的逻辑关系成立，输出结果为：

```
True
```

在第 4 行的代码中，2!=2 的逻辑关系不成立，输出结果为：

```
False
```

在第 5 行的代码中，6<>9 的逻辑关系成立，输出结果为：

```
True
```

2.5.3　基础知识——逻辑运算符与逻辑表达式

逻辑表达式就是用逻辑运算符和变量连接起来的式子。Python 语言中的逻辑运算符分为 3 种，分别是：逻辑与、逻辑或和逻辑非。C、Java 语言的逻辑运算符用&&、||、!来表示，Python 则采用 and、or 和 not 表示。表 2-4 列出了 Python 中的逻辑运算符和表达式。

表 2-4　Python中的逻辑运算符和表达式

逻辑运算符	逻辑表达式	描　述
and	x and y	逻辑与，当 x 为 True 时，才计算 y
or	x || y	逻辑或，当 x 为 False 时，才计算 y
not	not x	逻辑非

下面演示逻辑表达式的运算。

```
print 2 < 3 and 4 < 5
print 2 > 3 or 4 > 5
```

```
print not ( 2 < 3 and  4 < 5)
```

在第 1 行的代码中，2<3 的逻辑关系成立，并且 4<5 的逻辑关系也成立，因此使用 and 逻辑运算符连接两个成立的表达式之后，输出的结果为：

```
True
```

在第 2 行的代码中，2>3 的逻辑关系不成立，并且 4>5 的逻辑关系也不成立，因此使用 or 逻辑运算符连接两个不成立的表达式之后，输出的结果为：

```
False
```

在第 3 行的代码中，小括号里的表达式是成立的，即为 True。True 的逻辑非为 False，因此输出的结果为：

```
False
```

2.5.4　基础知识——运算符的优先级

Python 运算符在同一个表达式中的优先级是不同的。算术运算符的优先级大于关系运算符的优先级，关系运算符的优先级大于逻辑运算符的优先级，如图 2-5 所示。

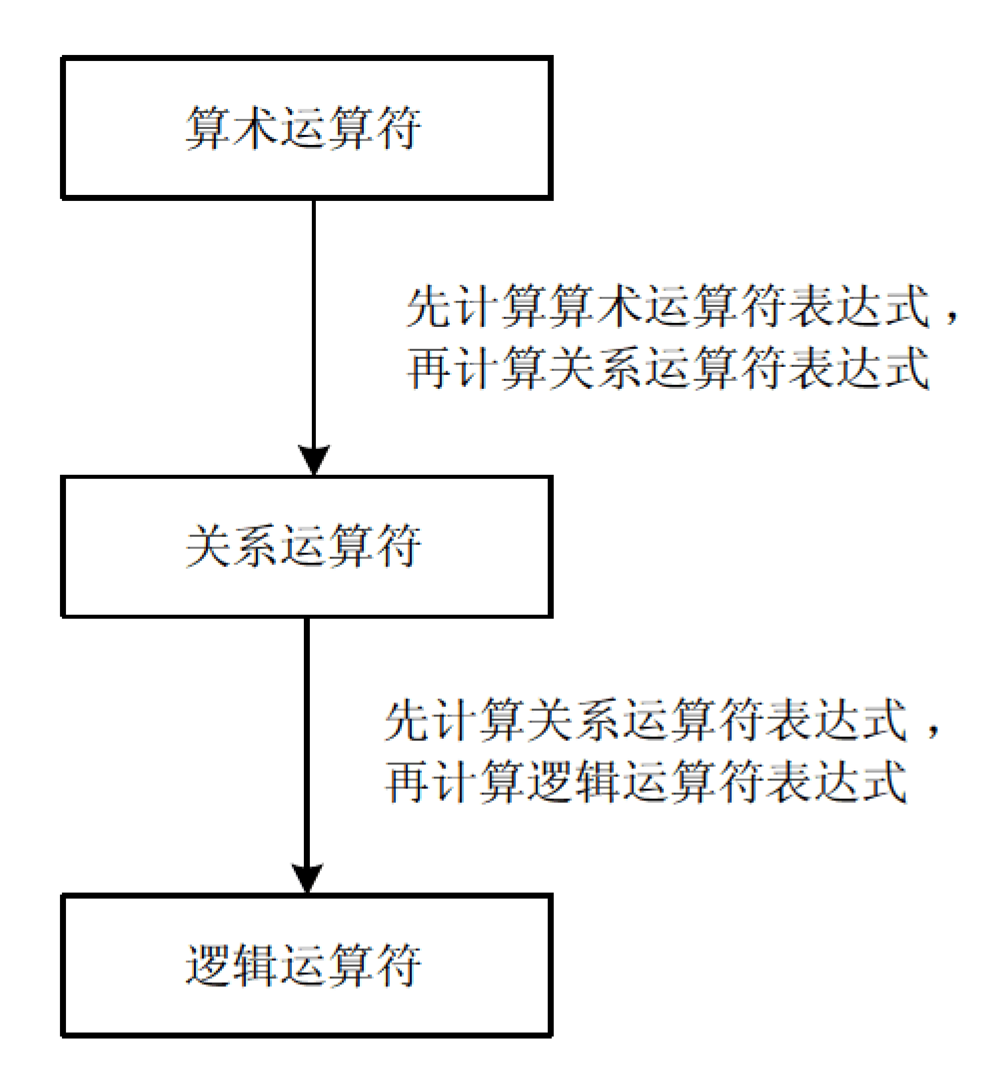


图 2-5　运算符的优先级

如果一个表达式中包含多种类型的运算符，那么 Python 会根据运算符的优先级从高到低进行计算。Python 中运算符的优先级从高到低的排列顺序如表 2-5 所示。



为了使代码具有更好的可读性，一般使用圆括号“()”来将表达式分组表示，例如 (3>2) && (5>4)。显而易见，系统会先计算圆括号中的表达式，然后再计算&&操作。

表 2-5　Python运算符的优先级排序

<table>
<tr><th>运 算 符</th><th>描　述</th></tr>
<tr><td>expression, ...</td><td>字符串转换</td></tr>
<tr><td>{key:datum, ...}</td><td>字典显示</td></tr>
<tr><td>[expression, ...]</td><td>列表显示</td></tr>
<tr><td>(expression , ...)</td><td>绑定或元组显示</td></tr>
<tr><td>f (arguments ...)</td><td>函数调用</td></tr>
<tr><td>x[index:index]</td><td>寻址段</td></tr>
<tr><td>x[index]</td><td>下标</td></tr>
<tr><td>x.attribute</td><td>属性</td></tr>
<tr><td>**</td><td>指数</td></tr>
<tr><td>−x</td><td>按位反转</td></tr>
<tr><td>+x，−x</td><td>正负号</td></tr>
<tr><td>*，/，%</td><td>乘、除、求余</td></tr>
<tr><td>+，−</td><td>加、减</td></tr>
<tr><td><<，>></td><td>移位</td></tr>
<tr><td>&</td><td>按位与</td></tr>
<tr><td>^</td><td>按位异或</td></tr>
<tr><td>|</td><td>按位或</td></tr>
<tr><td><，<=，>，>=，!=，==</td><td>比较运算符</td></tr>
<tr><td>is，is not</td><td>相同测试</td></tr>
<tr><td>in，not in</td><td>成员测试</td></tr>
<tr><td>not x</td><td>布尔非</td></tr>
<tr><td>and</td><td>布尔与</td></tr>
<tr><td>or</td><td>布尔或</td></tr>
<tr><td>lambda</td><td>Lambda 表达式</td></tr>
</table>

2.5.5　实例描述

正月十五闹元宵，很多城市以放烟花来庆贺这个节日，以安全为前提规定放烟花的范围为半径 1.5 米的圆。也就是说，在放烟花的时候，只能在这个半径为 1.5 米的圆内放置炮竹，然后才可以点燃。那么，这个圆有多大呢？下面来计算一下。

2.5.6　实例应用

【例 2-3】计算圆的周长和面积。

(1) 像 C 语言一样，等号用来给变量赋值，并且分配的值是只读的。在 Python 解释器中声明圆的半径变量为 r，并将 1.5 赋值给它，代码如下：

```
>>> r=1.5
```

(2) 圆的周长公式为 c=2∏R，其中∏的数值约为 3.14，R 表示的值是半径长 1.5 米。接下来继续在编辑器中输入如下代码：

```
>>> c=2*3.14*r
```

这个表达式表示将计算出的结果赋值给变量 c。其实，同一个值可以同时赋给几个变量的，例如：

```
>>> x=y=z=3
>>> x
3
>>> y
3
>>> z
3
```

(3) 将计算出来的圆周长输出。

```
>>> c
9.4199999999999999
```

(4) 接着编辑代码，计算圆的面积，公式为∏R^2，并将结果输出。

```
>>> s=3.14*(r**2)
>>> s
7.0650000000000004
```

2.5.7　运行结果

查看 Python 解释器的内容，如图 2-6 所示。

```
Python Shell
File  Edit  Shell  Debug  Options  Windows  Help
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.4
>>> r=1.5
>>> c=2*3.14*r
>>> c
9.4199999999999999
>>> s=3.14*(r**2)
>>> s
7.0650000000000004
>>> |
Ln: 19 Col: 4
```

图 2-6　计算圆的周长和面积的代码清单

2.5.8 实例分析



源码解析

通过上面的例子可以看出，Python 中的计算和其他语言中的计算是相同的。在上面的例子中，变量 r 的值为 1.5，即半径的长。在下面的计算中，如果使用了半径，直接调用 r 即可。当将计算结果赋值给变量 c 或者 s 后，在解释器中直接输入变量名称，即可输出变量的值。

2.6 常见问题解答

2.6.1 Python中 3 种字符串引号的区别



JDK 安装错误提示：The download file appears to be corrupted. Please refer to the Troubleshooting section of the Installation.

网络课堂：http://bbs.itzcn.com/thread-9851-1-1.html

我刚接触 Python 语言，在查找资料的时候，看到字符串很多都是用 3 个引号括起来的，这与两个引号或者单引号字符串有什么区别呢？

【解决办法】字符串可以用三重引号，例如"""或'''。三重引号中的字符串在行尾不需要换行标记，且所有的格式都会包括在字符串中。例如：

```
print """hao do you do?"""
print ''' I'm eating'''
```

运行该段代码，输出结果如下：

```
hao do you do?
I'm eating
```

解释器打印出来的字符串与它们输入的形式完全相同。

2.6.2 Python中文编码问题



Python 中文编码问题！

网络课堂：http://bbs.itzcn.com/thread-15814-1-1.html

今天闲来无事，试着使用 Python 语言开发一个商品信息管理系统，刚声明了一个变量，就报了一堆错误，真是郁闷啊！代码如下：

```
_proName='联想笔记本'
print _proName
```

运行代码出错，错误为：

```
SyntaxError: Non-ASCII character '\xc1' in file 11.py on line 1, but no encoding
declared; see http://www.python.org/peps/pep-0263.html for details
```

【解决办法】字符串在 Python 内部的表示是 Unicode 编码。在做编码转换时，通常需要以 Unicode 作为中间编码，即先将其他编码的字符串解码(decode)成 Unicode，再从 Unicode 编码(encode)成另一种编码。

decode()函数的作用就是将其他编码的字符串转换成 Unicode 编码，例如 str1.decode ('gb2312')，表示将 gb2312 编码的字符串 str1 转换成 Unicode 编码。encode()函数的作用是将 Unicode 编码转换成其他编码的字符串，例如 str2.encode('gb2312')，表示将 Unicode 编码的字符串 str2 转换成 gb2312 编码。转换时，一定要先搞明白字符串是什么编码，然后使用 decode()函数将字符串编码改为 Unicode 编码，然后再使用 encode()函数将编码改为其他编码格式。如果是在 UTF8 的文件中，该字符串就是 UTF8 编码；如果在 gb2312 编码的文件中，则其编码为 gb2312。

2.7　习　　题

一、填空题

(1) Python 使用符号__________表示注释。

(2) 可以使用__________符号把一行过长的 Python 语句分解成几行。

(3) __________是只能在函数或代码段内使用的变量，函数或代码段一旦结束，局部变量的生命周期也将结束，在函数或代码段外是调用不到的。

二、选择题

(1) 下列__________语句在 Python 中是非法的。

A. x = y = z = 1　　　　B. x = (y = z + 1)

C. x, y = y, x　　　　D. x += y

(2) 下列表达式的值为 True 的是__________。

A. 5+4j > 2-3j　　　　B. 3>2>2

C. (3,2)< ('a', 'b')　　　　D. 'abc' > 'xyz'

(3) 关于 Python 中的复数，下列说法错误的是__________。

A. 表示复数的语法是 real + image j

B. 实部和虚部都是浮点数

C. 虚部必须后缀 j，且必须是小写

D. 方法 conjugate 返回复数的共轭复数

三、上机练习

上机练习：实现一个计算器的功能

用户输入两个操作数，并输入一个算术运算符，根据用户输入的运算符(只能是+、-、*和

/)来操作两个数，最后计算出结果。运行结果如图 2-7 所示。

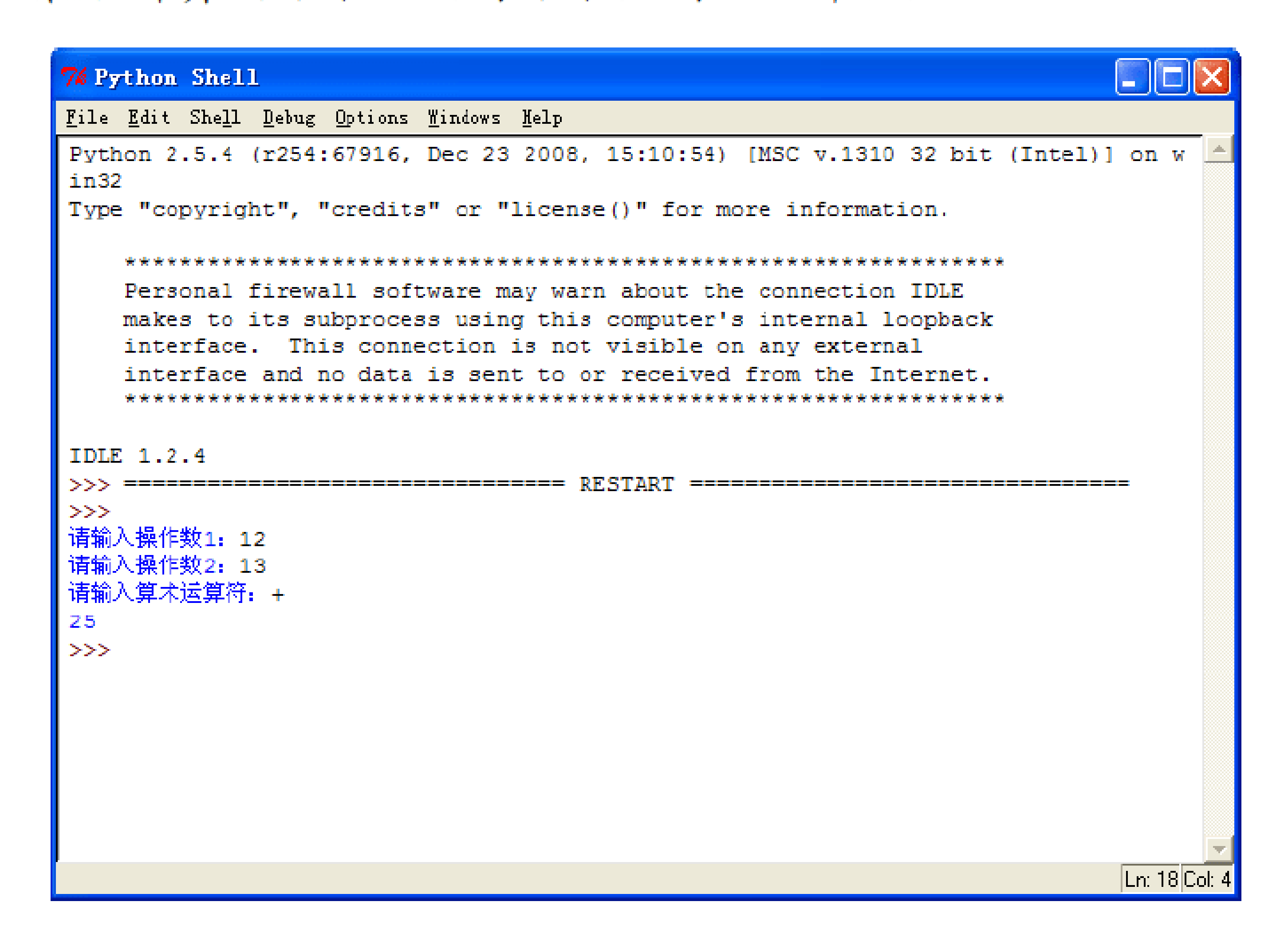


图 2-7　计算器的实现



第 3 章　控 制 结 构

内容摘要

首次接触 Python 语言，遇到需要使用循环来将程序打印输出的问题时，你有没有因为自己对 Python 语言的了解过少而懊丧万分？答案是肯定的。别着急，学了这一章之后，你就会有“柳暗花明又一村”的感觉。在本章中，将介绍 if 条件语句、while 循环、迭代器以及一些其他的迭代工具。同时还将介绍 break、continue 跳转语句，最后还将对一些不经常用的其他语句做简单的说明。

学习目标

- 掌握条件语句的使用。
- 熟练掌握循环语句以及迭代器的使用。
- 了解一些其他的迭代工具。
- 熟练使用循环中的跳转语句。
- 熟悉一些其他语句的用法。

3.1 制作有趣的炒菜流程

如果你做的菜淡了，那就再放一些盐。在生活中是不是经常听到类似的假设语句呢？在该语句中，使用了关联词“如果……就”。在编程世界里，能不能使用程序表达类似的关联词语句呢？答案是肯定的。在其他语言中，使用 if...else 的方式便可以将这条假设语句准确无误地输出，但在 Python 中如何表达呢？下面将详细介绍。

视频教学：光盘/videos/03/条件语句.avi　　长度：6 分钟

3.1.1 基础知识——条件语句

提及条件语句，对从事编程的人来说简直就是小菜一碟。使用最广泛的就是 if 语句，其作用主要是对表达式作判断，如果表达式为真，则返回一个代码块，否则返回另外一个代码块。那么在 Python 中条件语句如何使用呢？

如果需要在 Python 中熟练使用条件语句，那就必须了解真值。下面我们就来详细介绍一下。

1. 真值

真值也称为布尔值。在 Python 中，有一些值在作为布尔表达式时，会被解析器解析为 false，另外一些值则被解析成 true，如表 3-1 所示。

表 3-1　程序中的真值表

对象或者常量值	返 回 值
""	假
"string"	真
0	假
1	真
()空元素	假
[]空列表	假
{}空字典	假
None	假

下面通过一个例子来说明。

```
>>> bool('My name is dcy')
True
>>> bool(45)
True
>>> bool('')
False
>>> bool(0)
False
>>> bool(" ")
```

```
True
>>> bool("")
False
>>> bool([])
False
>>> bool(())
False
>>> bool({})
False
>>>
```

从上述代码可以看出，bool('My name is dcy')、bool(45)和 bool(" ")被解析后得到的值均为 True，而 bool('')、bool(0)、bool("")、bool([])、bool(())和 bool({})被解析后得到的值是 False。这说明标准值是 False、None、0、空字符串、空元素、空列表以及空字典时都为假，而其他的值都被解析成真。

在 Python 2.5 中，1 代表真，0 代表假，而 True 和 False 仅仅是 1 和 0 的一种唯美的修饰而已，外表不同，但实质是相同的。

下面的例子就是很好的证明，代码如下：

```
>>> True
True
>>> False
False
>>> False==0
True
>>> True==1
True
>>> True+False+12
13
>>>
```

此段代码说明了 True 和 False 分别代表 1 和 0。



在这里所有的值都可用作布尔值，不需要对它们进行显示转换。也就是说，Python 会自动转化这些值。

2. if语句

在不同的语言中，条件语句的使用语法格式不尽相同。使用 if 判断的条件语句，返回的结果是 True 或者 False，根据返回的结果来决定要执行的代码，常用于对数值或者表达式的比较运算。下面我们来看一下 if 语句在 Python 中的语法，格式如下：

```
if 表达式1 :
    代码块1
elif 表达式n:
    代码块n
else:
    代码块n+1
```

从上述语法可以看出，在 Python 语言中，if 语句块对应的是 elif 块，而在其他语言中 if 代码块对应的是 else if 或者其他。下面通过流程图来彻底解读一下代码的运行顺序，如图 3-1 所示。

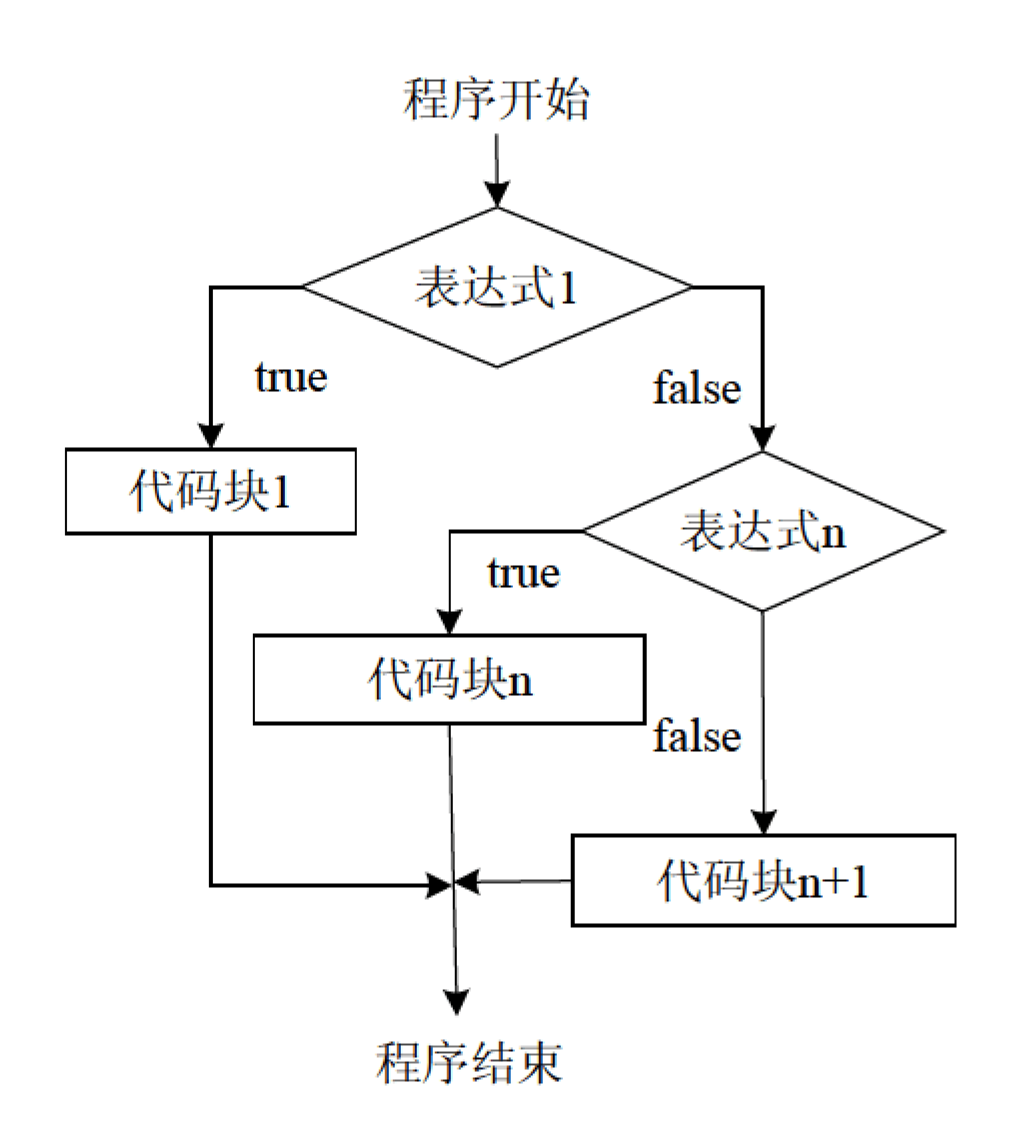


图 3-1　if语句的流程图

下面我们来看一个小例子，代码如下：

```
name='Happy'
if name=='Happy1':
    print 'My name is Happy1'
elif name=='Throw':
    print 'My name is Throw'
elif name=='Happy':
    print 'My name is Happy'
else:
    print 'sorry,no match'
```

在此段代码中，首先为 name 设置一个值，然后对 name 和==后面的值进行比较，若为 true，则输出相应的代码块。下面来看一下输出的结果。

```
My name is Happy
```



这里使用的 if...elif...else 的条件语句以及下面即将讲到的循环语句都需要将对应分句的缩进保持一致，否则在程序执行时会报错。

3.1.2　实例描述

你做过饭吗？假如你从来没有做过饭，但又想在朋友面前秀一下自己的厨艺。那么我们可以使用程序制作一个炒菜的过程，这样不仅能使我们记得下一步该干什么，而且还可以控制菜的咸淡程度。看到这里，有没有想试一下的冲动呢？

下面就来看一下实现方案吧。

3.1.3　实例应用

【例 3-1】制作有趣的炒菜流程。

(1) 创建一个名称为 ChaoCai.py 的文件。

(2) 在 ChaoCai.py 文件中添加如下代码：

```
initXQ='洗菜切菜等'
initJR='加热炒锅，锅干后，倒入适量的油'
hotting='用旺火翻炒，直到将菜炒热'
tiaoing='放入盐和味精等调料拌匀'
ts=6

if initXQ=='':
    print '洗菜切菜等'
elif initJR=='':
    print '加热炒锅，锅干后，倒入适量的油'
elif hotting=='':
    print '用旺火翻炒，直到将菜炒热'
elif tiaoing=='':
    print '放入盐和味精等调料拌匀'
if ts<5:
    print '放了 6 勺盐有点淡，再放一点盐'
else:
    print '将菜倒入盘中，端到客厅'
```

(3) 保存修改好的代码。

3.1.4　运行结果

执行程序，运行结果如下：

```
将菜倒入盘中，端到客厅
```

3.1.5　实例分析



源码解析

在本实例中，首先使用 initXQ==:的方式判断 initXQ 是否为空，如果不为空，则进行下一步选择。在这里我们仅仅使用了 if...elif...else 条件语句做判断。

3.2　九九乘法表

曾听人说：在这个充满技术气息的年代里，什么人最可怕，高级程序员最可怕。正是这句话激起了我对程序的兴趣，这也是我选择程序的一个重要原因。例如：使用程序中的循环语句可以简单地将所有考生的成绩输出，这样既省时又省力。使用循环语句也可以使乘法口诀表有规律地显示出来。

下面我们来看一下在 Python 语言中，如何使用循环语句实现九九乘法表的打印输出。

 **视频教学：**光盘/videos/03/循环语句.avi　　　　长度：10 分钟

3.2.1　基础知识——循环语句

循环，这个与我们生活密切相关的词语，在我们的生活中无时无刻不在上演。例如：我们喜欢某首歌曲，就可以将该首歌设置为循环播放模式；若不发生特殊状况，每天上下班的路线，何尝不是另一种形式的循环呢！下面我们来看一下在编程语言中如何实现循环。

在 Python 语言中，循环语句主要有两种，分别是 while 循环和 for 循环。下面首先来看 while 循环。

1. while循环

众所周知，while 循环是当指定的条件为 true 时，循环执行 while 块中的语句。在 Python 语言中，while 语句的语法格式如下：

```
while 条件:
    代码块 1
else
    代码块 2
```

如果你仍然感觉有些迷惑，那么看看它的流程图将会让你豁然开朗，如图 3-2 所示。

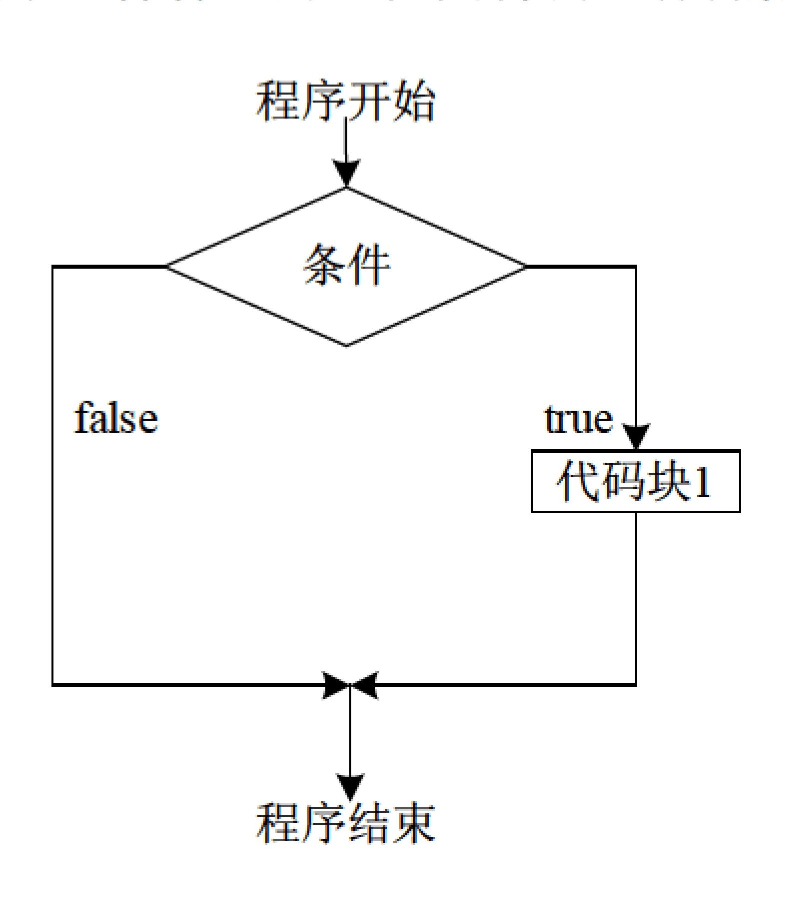


图 3-2　while循环语句的流程图

接下来，我们通过一个小例子来说明一下。

```
i=1
while i<=5:
    print '我是第'+str(i)+'次输出'
    i+=1
else:
    print '输出结果完毕'
```

在该段代码中，首先声明了变量 i=1，接着进行条件判断 i<=5，符合条件就打印输出 while 中的代码块 5 次。下面来看一下输出的结果是不是和我们推测的一样。运行程序，输出的结果如下：

```
我是第 1 次输出
我是第 2 次输出
我是第 3 次输出
我是第 4 次输出
我是第 5 次输出
输出结果完毕
```

现在，是不是很清晰了。

2. for循环

通过上面的介绍，我们了解到 while 语句可以用来在任何条件为真的情况下重复执行一个代码块，这说明 while 语句非常灵活。但是如果要让一个集合或者字典中的每个元素都执行一个代码块，使用 while 循环是不是有些难度了？这时候我们使用 for 循环就可以轻而易举地解决问题。

下面我们来看一下 for 循环的语法格式。

```
for target in object:
    代码块 1
    if 条件 1:
        break
    if 条件 2:
        continue
else:
    代码块 2
```

下面通过例子来说明 for 循环的使用。

例如：我最近特别喜欢一首诗《错误》，作者郑愁予。因此我想将这些诗句存放到列表里面，然后使用 for…in 循环将诗句遍历输出。这样我既能学到知识又能记住优美的诗句，一举两得。代码如下：

```
yiqie=[
'错误',
'郑愁予',
'我打江南走过',
'那等在季节里的容颜如莲花的开落',
'东风不来，三月的柳絮不飞',
'你的心如小小寂寞的城',
'恰若青石的街道向晚',
'音不响，三月的春帷不揭',
```

```
'你的心是小小的窗扉紧掩',
'我达达的马蹄是美丽的错误 ',
'我不是归人，是个过客……']
for nei in yiqie :
    print nei
```

保存代码之后，运行程序，其结果如下：

```
错误
郑愁予
我打江南走过
那等在季节里的容颜如莲花的开落
东风不来，三月的柳絮不飞
你的心如小小寂寞的城
恰若青石的街道向晚
音不响，三月的春帷不揭
你的心是小小的窗扉紧掩
我达达的马蹄是美丽的错误
我不是归人，是个过客……
```

从上述结果可以看出，我们使用 for…in 循环语句来遍历一个集合 yiqie。接下来介绍如何循环遍历字典元素。

所谓字典就是 Python 中重要的数据类型，由“键-值”组成的集合。我们只要使用一个简单的 for 循环就能遍历字典中的所有键和值，很神奇吧！

下面再来看一个小例子，代码如下：

```
meinv={
'赵飞燕':'水色箫前流玉霜，赵家飞燕侍昭阳，掌中舞罢箫声绝，三十六宫秋夜长。',
'王昭君':'落燕北飞情未消，折戟沉沙和两朝。可怜琵琶空对月，万般愁情镜中描。',
'西施  ':'一代倾城逐浪花，吴宫空自忆儿家，效颦莫笑东村女，头白溪边尚浣沙。',
'貂蝉  ':'榆钱不买千金笑，柳带何须百宝妆。舞罢隔帘偷目送，不知谁是楚襄王。'}
for key in meinv:
    print key,'被形容的诗句是',meinv[key]
```

保存代码，运行程序，显示的结果如下：

```
王昭君 被形容的诗句是 落燕北飞情未消，折戟沉沙和两朝。可怜琵琶空对月，万般愁情镜中描。
西施   被形容的诗句是 一代倾城逐浪花，吴宫空自忆儿家，效颦莫笑东村女，头白溪边尚浣沙。
貂蝉   被形容的诗句是 榆钱不买千金笑，柳带何须百宝妆。舞罢隔帘偷目送，不知谁是楚襄王。
赵飞燕 被形容的诗句是 水色箫前流玉霜，赵家飞燕侍昭阳，掌中舞罢箫声绝，三十六宫秋夜长。
```

从上述结果来看，我们将名称“赵飞燕”等作为键，而描写她们的诗句作为值存放到名称为 meinv 的字典里，然后通过 for 语句将存入的键输出，接着使用 meinv[key]的方式将存入到字典中的值输出。这样，字典中所有的键和值都会被输出了。

上面介绍了如何使用 for 循环对列表和字典进行遍历输出，由于任何序列都适用 for 循环，因此它是通用的工具。下面就来看看如何对字符串和元祖进行循环。

针对字符串的循环，下面仍然通过一个小实例来说明。

```
zifu='yang'
for zi in zifu:
    print zi
```

保存代码，执行的结果如下：

```
y
a
n
g
```

从上述结果可以看出，我们使用 for … in 的方式将字符串 yang 以字符的形式循环遍历。

接下来，我们再看一下使用 for 循环遍历元祖。代码如下：

```
shuzu=[(1,2),(3,4),(5,6)]
for (a,b) in shuzu:
    print a,b
```

在上述代码中，使用 for…in 循环将元祖 shuzu 遍历，接着打印输出 a 和 b 的值。此时在关键字 for 后面的(a,b)类似于元祖中的(1,2)等元素，那么首次被打印的元素应该是 1 和 2。保存代码，然后看看程序运行的结果。

```
1 2
3 4
5 6
```

从上述结果可以看出，第一次经过循环就像是在编写(a,b)=(1,2)，而第二次就像是将(3,4)对应赋值给(a,b)，依次类推。这是怎么回事？下面我们将该例子修改一下，以证明是不是特殊状况，修改的代码如下：

```
shuzu=[(1,2),(3,4),(5,6)]
for a in shuzu:
    print a
```

在该段代码中，我们将 for 后面的关键字(a,b)换成 a，那么首次打印的结果是不是(1,2)呢？保存代码，运行程序并查看其结果。

```
(1, 2)
(3, 4)
(5, 6)
```

从上述结果可以看出，我们的推测是正确的。可以证明这种类似赋值的方式不是一种特殊的状况，可见任何一种类似赋值运算在语法上都能用在关键字 for 之后。

3. 迭代器

我们已经提到，for 循环可以用于 Python 中任何序列类型，例如列表、元祖、字典以及字符串。另外，for 循环还可以用于任何可迭代的对象。可迭代对象可以说是在迭代工具环境中一次产生的对象，包括实际序列和按照需求而计算出的虚拟序列。那么哪些可以称之为迭代工具呢？类似于 for 循环，可以从左到右进行扫描的工具均可称为迭代工具。下面来看一下迭代器，首先来看一下文件迭代器。

对于已经打开的文件对象，它便具有两个方法 readline 和 next。其中 readline 方法是一次从文件中读取一行文本，当调用 readline 方法时，就会前进到下一列，直到文件末尾，返回一个空字符串。对于 next：方法当执行到文件末尾时，不是返回空字符串，而是引发内置的 StopIteration 异常。下面通过一个小例子来说明。

(1) 创建一个名称为 ceshi.py 的文件，并添加如下代码：

```
dream=raw_input('输入您未来的梦想')
if dream!='':
    print '我最大的梦想就是',dream
```

(2) 创建一个名称为 file.py 的文件，并添加如下代码：

```
f=open('ceshi.py')
print f.readline()
print f.readline()
print f.readline()
print f.readline()
```

在上述代码中，我们使用了 open 方法来打开 ceshi.py 文件对象，然后使用文件的 readline 方法将文本读出。在此期间，我们调用了 4 次 readline 方法，而文件 ceshi.py 只有 3 行代码，会出现什么样的情况呢？看一下程序运行结果。

```
dream=raw_input('输入您未来的梦想')
if dream!='':
    print '我最大的梦想就是',dream
```

从上述结果来看，并没有出现什么错误，说明文件的 readline 方法读到文本的末尾行返回的是空字符串。而 next 方法就不一样了，再来看一下。

(3) 创建一个名称为 filenext.py 的文件，并添加如下代码：

```
f=open('ceshi.py')
print f.next()
print f.next()
print f.next()
print f.next()
```

(4) 保存代码，查看其执行结果。

```
dream=raw_input('输入您未来的梦想')
if dream!='':
    print '我最大的梦想就是',dream
Traceback (most recent call last):
  File "file.py", line 5, in <module>
    print f.next()
StopIteration
```

正如上述结果所示，文本末尾出现了 StopIteration 异常。

Python 中所谓的迭代协议就是 next 方法的对象会前进到下一个结果，在一系列结果的末尾会引发 StopIteration 异常。在 Python 中，任何类似的对象都被认为是可迭代的。由于所有迭代工具的内部工作都是在循环调用 next 方法，并且捕捉 StopIteration 异常来确定何时离开，因此我们可以使用 for 循环或者其他工具来遍历这类对象。

下面就来看一下如何使用 for 循环将文件中的文本行迭代输出，代码如下：

创建一个名为 filefor.py 的文件，并添加如下代码：

```
for redline in open('ceshi.py'):
    print redline
```

运行程序，其执行结果如下：

```
dream=raw_input('输入您未来的梦想')
if dream!='':
```

```
    print '我最大的梦想就是',dream
```

通过上面的介绍，我们了解了什么是文件迭代器。接下来要讲的是使用 iter()内置函数可以返回迭代器，然后使用迭代工具将其输出。下面通过一个例子来说明。代码如下：

```
mys=['贵妃醉酒','貂蝉拜月','西施浣纱','昭君出塞']
my=iter(mys)
print my.next()
print my.next()
print my.next()
print my.next()
```

执行结果如下：

```
>>>
贵妃醉酒
貂蝉拜月
西施浣纱
昭君出塞
>>>
```

从运行结果可以看出，在代码中我们使用 iter()函数返回一个迭代器，然后使用迭代器的 next 方法将之输出。

当然，我们还可以使用迭代工具 for 循环，例如：

```
mys=['贵妃醉酒','貂蝉拜月','西施浣纱','昭君出塞']
for my in iter(mys):
    print my
```

运行得出的结果和使用迭代器的 next 方法得出的结果是一样的。

3.2.2　实例描述

“你会默写九九乘法表吗？”我那可爱的小侄女问我。她得到的答案当然是肯定的。只是当时正在接触 Python 语言的我陷入了沉思，我是一个理想主义者，对自己的事讲求精益求精。使用 Python 语言将九九乘法表按规范输出，虽然想到了使用 for 循环，但是由于对其格式不太了解，因此迟迟没有下手。正巧，通过对这一节的学习，我感觉实现九九乘法表简直是小菜一碟。下面来看一下我的实现思路。

3.2.3 实例应用

【例 3-2】九九乘法表。

(1) 创建一个名称为 chengfa.py 的文件。

(2) 在 chengfa.py 文件中添加如下代码：

```
for i in range(1,10):
    for j in range(1,i+1):
        print(" " .join(["%d*%d=%d" %(j,i,i*j)])
```

(3) 保存代码。

3.2.4 运行结果

运行程序，得到如下结果：

```
>>>
1*1=1
1*2=2 2*2=4
1*3=3 2*3=6 3*3=9
1*4=4 2*4=8 3*4=12 4*4=16
1*5=5 2*5=10 3*5=15 4*5=20 5*5=25
1*6=6 2*6=12 3*6=18 4*6=24 5*6=30 6*6=36
1*7=7 2*7=14 3*7=21 4*7=28 5*7=35 6*7=42 7*7=49
1*8=8 2*8=16 3*8=24 4*8=32 5*8=40 6*8=48 7*8=56 8*8=64
1*9=9 2*9=18 3*9=27 4*9=36 5*9=45 6*9=54 7*9=63 8*9=72 9*9=81
>>>
```

3.2.5 实例分析

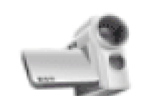


源码解析

在本实例中，我们使用了两个嵌套的 for 循环，其中外层 for 循环是循环的行，而内层则是循环的列。首次循环 i 的值是 1，j 的值也是 1。也就是说，第一行是一行一列，即打印的结果是 1*1=1，依此类推就可以将九九乘法表打印出来。很简单吧。

3.3 实现关键字搜索功能

当你进入一个购物网站时，如果在首页没有找到你想要的商品，你会怎么办？恐怕大多数人都会想到，根据关键字进行搜索，之后所有和你输入的关键字相关的商品都会显示出来。这样既省时又省力。在 Python 语言中，能否实现这样的功能呢？学了本节之后，你就会惊讶地说：真的很神奇啊，使用迭代工具也能实现关键字搜索！话不多说，赶快往下看吧。

视频教学：光盘/videos/03/迭代工具.avi　　　　长度：5 分钟

3.3.1　基础知识——迭代工具

在前面的章节中，我们提到 for 循环是最常用的迭代工具。本节主要介绍迭代的几种方式，例如并行迭代、编号迭代等。

1．并行迭代

并行迭代即程序可以同时迭代两个程序。例如：有两个列表分别是姓名和年龄，结果需要打印出姓名和对应的年龄。代码如下：

```
names=['duanchunyang','maxianglin','lintiantian','ltt','dcyandly']
ages=[23,22,21,25,26]
for i in range(len(names)) :
    print names[i],'的年龄是',ages[i]
```

在上述代码中，i 是循环索引的标准变量名。

上面我们使用 range()函数对两个列表进行迭代，另外还可以使用内建的函数 zip 进行并行迭代。zip 迭代是把两个序列“压缩”在一起，然后返回一个元祖的列表。使用 range()函数的代码如下：

```
names=['duanchunyang','maxianglin','lintiantian','ltt','dcyandly']
ages=[23,22,21,25,26]
for name,age in zip(names,ages) :
    print name,'的年龄是',age
```

保存代码。运行程序，执行结果如下所示：

```
>>>
duanchunyang 的年龄是 23
maxianglin 的年龄是 22
lintiantian 的年龄是 21
ltt 的年龄是 25
dcyandly 的年龄是 26
>>>
```

接下来，我们看一下编号迭代。

2．编号迭代

在一个文档中，如果我们想替换一个词，通常会按 Ctrl+F 组合键将该词选中，然后在替换框里填写要替换的词，接着单击“全部替换”按钮，这样文档中的词就全部被替换了。这样的情况，在程序中能不能实现呢？答案是肯定的。在程序中使用编号迭代的方式不仅能获得序列中的对象，还能获取当前对象的索引。下面我们就来看一下如何使用编号迭代。

例如有一个列表，列表中存放了一组诗句。我想在该组诗句中替换所有包含某个字的诗句，代码如下所示：

```
shijus=['星桥鹊驾，','经年才见，','想离情、','别恨难穷。']
for index,shiju in enumerate(shijus):
    if '星桥鹊驾' in shiju:
        shijus[index]='金风玉露一相逢，便胜却人间无数。'
        print index,shiju
        for shi in shijus:
            print shi
```

在上述代码中，我们使用了内建的 enumerate 函数来进行编号迭代，其作用主要是在提供索引的地方迭代索引值对。

运行程序，执行结果如下所示：

```
>>>
0 星桥鹊驾，
金风玉露一相逢，便胜却人间无数。
经年才见，
想离情、
别恨难穷。
>>>
```

3.3.2 实例描述

通过本节的学习，我感觉编号迭代挺神奇的。我想为自己做一个鲜花名称关键字搜索的功能，以列表作为数据库，以可以输入字符的平台作为接收前台来达到搜索的目的。下面就是我的实现方案。

3.3.3 实例应用

【例 3-3】实现关键字搜索功能。

(1) 创建一个名称为 sousuo.py 的文件。

(2) 在 sousuo.py 的文件中添加如下代码：

```
zifu=raw_input('输入您要查询鲜花的名称：')
shujus=['长春花','珍珠花','向日葵','风铃草','金盏菊','含羞草','夹竹桃','大丽花','金雀花','野蔷薇','桔梗花']
for index,shuju in enumerate(shujus):
    if zifu in shuju:
        print shuju
```

(3) 保存修改好的代码。

3.3.4 运行结果

运行程序，当输入关键字“花”时，运行结果如下：

```
>>>
输入您要查询鲜花的名称：花
长春花
珍珠花
大丽花
金雀花
桔梗花
>>>
```

3.3.5　实例分析



源码解析

在本实例中，我们主要使用了 enumerate 函数来返回一个迭代器，然后使用 for 循环将列表中的元素迭代输出，接着使用 if 条件语句，之后使用了 in 关键字来判断该字符是否被列表中的元素包含。如果被包含，则输出该元素，否则不输出任何字符。

3.4　为歌曲列表制作新颖的循环模式

听音乐不仅能放松心情，而且能陶冶情操，可谓是心情烦躁时的良药啊。听一些优美的音乐，我们一般喜欢将模式调到单曲循环模式，即使是天籁之声也容不得我们听百遍啊。试想一下，如果将你喜欢的某些曲目放到一个列表中，然后循环列表中的歌曲，倘若你厌倦了该列表中的某一首歌曲，该模式可以将此首歌跳过继续播放下一首歌曲。这样是不是一个很不错的选择呢？

下面就让我们来学习如何制作这种歌曲播放模式。



视频教学：光盘/videos/03/跳转语句.avi　　　长度：7 分钟

3.4.1　基础知识——跳转语句

在使用 while 或者 for 循环时，当执行的条件为假或者序列元素被完全遍历时，会跳出循环。但是有时候我们想提前中断该循环，然后进入新的迭代，或者结束该循环，此时跳转语句无疑是最好的选择。

在 Python 语言中，常用的结束一个循环的语句有两种，分别是：break 语句和 continue 语句。首先来看一下 break 语句。

1. break语句

提到 break 语句，我想大家并不陌生。在程序中，如果我们遇到死循环的状况，break 语句恐怕是首要之选。在一个循环语句中出现 break 关键字，表示跳出整个循环，如果在嵌套循环中出现 break 关键字，则表示跳出此嵌套循环。

下面通过一个例子来说明 break 语句的使用方法。代码如下：

```
i=1
while i<=5:
    print i
    if i==4:
        print '打印到此停止的数是',i
        break
    i+=1
```

在此段代码中，我们将条件为 i<=5 的数循环输出，等到 i 的值为 4 时能否跳出整个循环呢？接下来，我们看一下执行的结果。

```
1
2
3
4
打印到此停止的数是 4
```

从上述结果可以看出，当打印 i 的值为 4 时就跳出整个循环。如果将 break 关键字去掉，执行的结果如下：

```
1
2
3
4
打印到此停止的数是 4
5
```

接着我们来看一个在嵌套循环语句中使用 break 关键字的例子。代码如下：

```
items=['my name is duanchunyang','I miss you',(4,5),2]
keys=['my name is duanchunyang',(4,5),'I miss you','I am tired']
for key in keys :
    for item in items :
        if item==key:
            print key,'was found'
            break
    else:
        print key,'not found'
```

在上述代码中，使用了两个 for 循环，并且两个循环在同时运行：外层循环扫描键列表，内层循环为每个键扫描元素列表。循环中的 else 分句的嵌套是很关键的，其缩进至和内层 for 循环首行相同层次，因此是和内层循环相关联的。break 关键字在嵌套的循环中出现，如果符合条件那么跳出的是内循环。下面来看一下执行的结果。

```
my name is duanchunyang was found
(4, 5) was found
I miss you was found
I am tired not found
```

从上述结果可以看出，这里嵌套的 if 会在找到与之相符合的结果时执行 break 语句，而循环中的 else 分句认定如果执行此处，意味着搜索失败。

在 Python 中，while 和 for 循环非常灵活，只是一旦使用 while 语句就会遇到一个需要更多功能的问题。例如：用户在提示符下输入文字时需要执行的语句，或者用户不输入任何字符时

停止循环。我们可以使用下面的方法来写代码。

```
word='dummy'
while word:
    word=raw_input('请输入您的用户名')
    print '您输入的用户名是 '+word
```

执行结果如下所示：

```
>>>
请输入您的用户名 dcy
您输入的用户名是 dcy
请输入您的用户名 luying
您输入的用户名是 luying
请输入您的用户名 lwsd
您输入的用户名是 lwsd
请输入您的用户名
您输入的用户名是
>>>
```

从上述结果来看，当程序没有进入循环之前，我们为 word 赋了一个未使用的值，称为哑值。使用哑值就是工作没有尽善尽美的标志，我们可以使用另外一种赋值方式，从而避免使用哑值。下面让我们来看一下修改之后的代码。

```
word=raw_input('请输入您的用户名')
while word:
    print '您输入的用户名是 '+word
    word=raw_input('请输入您的用户名')
```

在上述代码中，我们没有使用哑值，但是有了重复的代码，那就是在两个地方使用了同样的赋值语句并且两次调用 raw_input。这样显然不好，那么有没有其他办法呢？当然有，那就是使用 while True/break 语句。下面介绍一下 while True/break 语句。

while True 语句实现了一个永远不会自己停止的循环。如果想要终止循环，则需要在循环的内部添加 if 语句，当条件满足时调用 break 语句。

下面通过一个小例子来说明，代码如下：

```
while True:
    word=raw_input('输入您的用户名')
    if not word:
        break
        print '输入的用户名是 '+word
```

在上述代码中，我们使用了 while True 语句，然后在 if 后面作判断，如果符合条件就执行 break 语句。

运行程序，执行结果如下：

```
>>>
输入您的用户名 duanchunyang
输入您的用户名 dcy
输入您的用户名 admin
```

```
输入您的用户名 ly
输入您的用户名 ldx
输入您的用户名
>>>
```

从上述结果可以看出，当我们不输入任何字符而直接敲回车键时，执行结果是跳出整个循环。

2. continue语句

continue 语句相对于 break 语句来说用得比较少。在程序中的意思是结束当前的迭代，立即跳到循环的顶端，也就是"跳过剩余的循环体，但不是结束该循环"。如果你想跳过循环体的某个部分，那么 continue 就是很好的选择。

下面通过一个小例子来说明 continue 的使用方法，代码如下：

```
i=1
while i<=5:
    name=raw_input('输入您的用户名')
    age=raw_input('输入您的年龄')
    if name=='dcy':
        continue
    print 'hello','您输入的名字是',name,'您输入的年龄是',age
```

在上述代码中，我们通过使用 while 循环为用户设定了 5 次输入机会，然后通过使用函数 raw_input 实现用户手动输入名字和年龄。接着判断 if 后面的条件，如果条件符合就执行 continue 语句，否则执行打印输出的语句。下面来看一下执行的结果。

```
输入您的用户名 dcy
输入您的年龄 20
输入您的用户名 accp
输入您的年龄 20
hello 您输入的名字是 accp 您输入的年龄是 20
```

从上述结果可以看，第一次用户输入的用户名是 dcy，符合条件，程序执行 continue 语句，跳转到循环的顶端，因此没有打印输出。第二次用户输入的用户名是 accp，不符合 if 后面的条件，故执行打印输出的语句。

3.4.2 实例描述

我是一个上班族，如果要加上一个修饰语的话，那就是经济条件很不好的上班族。和大多数人相同的是：每天都得挤公交。不同的是：我每天都戴着耳机，沉湎于我的音乐世界中。但我唯一不满意的就是我的手机只有循环播放当前歌曲、全部歌曲以及随机播放这几种模式。我特想把所有想听的歌曲放到一个播放列表中，然后一次听个够，如果在此期间对某首歌曲厌倦了，就输入该歌曲的名称，使其在播放的时候跳过。

由于经济条件不允许，在现实世界里没有办法实现，那么只有在虚拟的程序世界中过一下瘾了。下面来看一下我的实现方案。

3.4.3　实例应用

【例 3-4】为歌曲列表制作新颖的循环模式。

(1) 创建一个名称为 gequ.py 的文件。

(2) 在 gequ.py 的文件中添加如下代码。

```
gequ=['爱，就是爱','全面通缉','离开那天','明天过后','见或者不见','莫失莫忘']
countStr=raw_input('你想让音乐循环播放的遍数：')
count=int(countStr)
qizhong=raw_input('输入您目前不想听的歌曲：')
tiaoguo=raw_input('输入您想跳过的歌曲：')
i=1
while i<=count:
    i+=1
    print '---------------循环开始--------------'
    for danqu in gequ:
        if danqu==qizhong:
            break
        if danqu==tiaoguo:
            continue
        print '第',i-1,'次播放的歌曲',danqu
```

(3) 保存修改好的代码。

3.4.4　运行结果

运行程序。当输入要循环遍数为 2 时，执行结果如下所示：

```
>>>
你想让音乐循环播放的遍数：2
输入您目前不想听的歌曲：
输入您想跳过的歌曲：
---------------循环开始--------------
第 1 次播放的歌曲 爱，就是爱
第 1 次播放的歌曲 全面通缉
第 1 次播放的歌曲 离开那天
第 1 次播放的歌曲 明天过后
第 1 次播放的歌曲 见或者不见
第 1 次播放的歌曲 莫失莫忘
---------------循环开始--------------
第 2 次播放的歌曲 爱，就是爱
第 2 次播放的歌曲 全面通缉
第 2 次播放的歌曲 离开那天
第 2 次播放的歌曲 明天过后
第 2 次播放的歌曲 见或者不见
第 2 次播放的歌曲 莫失莫忘
>>>
```

当输入要循环的遍数为 2，且不想听的歌曲是“明天过后”，想要跳过的歌曲是“全面通

缉”时，执行结果如下：

```
>>>
你想让音乐循环播放的遍数：2
输入您目前不想听的歌曲：明天过后
输入您想跳过的歌曲：全面通缉
---------------循环开始--------------
第 1 次播放的歌曲 爱，就是爱
第 1 次播放的歌曲 离开那天
---------------循环开始--------------
第 2 次播放的歌曲 爱，就是爱
第 2 次播放的歌曲 离开那天
>>>
```

当输入要循环的遍数为 2，且不想听的歌曲是“全面通缉”时，执行结果如下所示：

```
>>>
你想让音乐循环播放的遍数：2
输入您目前不想听的歌曲：全面通缉
输入您想跳过的歌曲：
---------------循环开始--------------
第 1 次播放的歌曲 爱，就是爱
---------------循环开始--------------
第 2 次播放的歌曲 爱，就是爱
>>>
```

3.4.5 实例分析


源码解析

在本实例中，主要使用了 break 语句和 continue 语句。其中 break 语句在符合条件的情况下跳出整个循环，而 continue 语句则是跳过符合条件的曲目，然后跳转到 while 的顶端继续循环。

3.5 其 他 语 句

前面我们介绍了 if 条件语句、while 和 for 循环语句以及跳转语句，接下来看一下其他语句，例如 pass、del 和 exec 语句。首先来看一下 pass 语句。


视频教学：光盘/videos/03/其他语句.avi　　长度：6 分钟

3.5.1　基础知识——pass语句

pass 在程序中什么事情都不需要做，类似于一个占位符。是不是感觉很奇怪？俗话说，“不怕一万就怕万一”，类似这种什么事情都不需要做的状况一旦出现，非 pass 莫属。例如：我们有一个条件语句的例子，有很多需要判断的条件，但是具体执行的代码模块还有待补充，那么这个需要补充的模块就可以使用 pass 语句。代码如下：

```
name=raw_input('输入您的用户名：')
if name=='duanchunyang':
    print '您是管理员，请进'
elif name=='maxianglin':
    #对不起，还没有给您分配身份，请稍等一下
    pass
elif name=='lintiantian':
       #对不起，还没有给您分配身份，请稍等一下
    pass
```

运行程序。当输入的用户名为 duanchunyang 时，执行结果如下：

```
>>>
输入您的用户名：duanchunyang
您是管理员，请进
>>>
```

当输入的用户名为 maxianglin 时，执行结果如下：

```
>>>
输入您的用户名：maxianglin
>>>
```

3.5.2　基础知识——del语句

一般来说，Python 会删除那些不再使用的对象，因为使用者不会再通过任何变量或者数据结构来引用它们。那么在 Python 中如何删除那些不再使用的对象呢？很简单，在前面的章节中我们讲到使用 del 语句来删除序列和字典元素，在这里同样适用。使用 del 语句不仅会移除对一个对象的引用，也会移除这个名字本身。下面通过一个例子来说明，代码如下：

```
name='duanchunyang'
del name
print name
```

在上述代码中，我们使用 del 语句将名字 name 移除，接着查看运行的结果。

```
Traceback (most recent call last):
  File "pass1.py", line 3, in <module>
    print name
NameError: name 'name' is not defined
```

从上述结果报出错误可以得知，名称 name 已经被移除了，因此才会出现'name' is not defined 这样的错误。

上面我们提到使用 del 语句既可以移除对象的引用，也可以移除名字本身。那么如果是两

个名称都指向同一个值，会出现什么样的效果呢？下面我们看一下如下代码。

```
names='duanchunyang'
keys=names
del names
print keys
```

在上述代码中，我们使用 del 语句将名称 names 移除了，是不是就说明 keys 也跟着移除了呢？再来看一下执行的结果。

```
duanchunyang
```

从上述结果可以看出，将names移除之后并不影响keys，为什么会这样呢？原因是在Python中删除的只是名称，而非值本身。

3.5.3 基础知识——exec语句

exec 语句用来动态地创造 Python 代码，然后将其作为语句执行。

例如输出一个字符串，代码如下：

```
exec "print   '你好几次问我,那是什么？这就是爱'"
```

执行结果如下：

```
>>>
你好几次问我,那是什么？这就是爱
>>>
```



如果程序将用户提供的内容作为代码执行，很可能就会失去对代码执行的控制。在用之前一定要考虑清楚。

3.6 常见问题解答

3.6.1 Python中语句嵌套问题



Python 中语句嵌套问题。

网络课堂：http://bbs.itzcn.com/thread-12373-1-1.html

我刚学习 Python 语言，我想知道在 Python 中，for 循环可不可以和 while 循环嵌套使用？

【解决办法】当然可以，所有的编程语言都支持嵌套循环。例如：

```
for i in range(1, 5) :
        j = 1
        while j < 3 :
                print i * j
                j += 1
```

其执行结果如下：

```
1
2
2
4
3
6
4
8
```

3.6.2　Python中语句缩进问题



Python 中语句缩进问题。
网络课堂：http://bbs.itzcn.com/thread-12373-1-1.html

我刚学习 Python 语言，在编辑器里面运行以下代码：

```
length = 5
breadth = 2
area = length * breadth
print 'Area is', area
print 'Perimeter is', 2 * (length + breadth)
```

运行完毕后，出现以下错误提示，请问该怎么解决？

```
  File "wenti1.py", line 1
    length = 5
    ^
IndentationError: unexpected indent
```

怎么会出现这样的提示呢？

【解决办法】在 Python 语言中，需要注意语句的缩进，否则就会引发错误。上面这段代码就是因为语句缩进不当而引发的错误。如果你在 Python 2.5.4 编辑器中，编辑如下代码：

```
>>> length=5
>>> breadth=2
>>> area=length*breadth
>>> print area
10
```

也可以这样写：

```
length=5
breadth=2
area=length*breadth
print 'Area is',area
print 'Perimeter is',2*(length*breadth)
```

保存后运行程序，执行结果如下：

```
>>>
Area is 10
Perimeter is 20
>>>
```

3.6.3 Python中循环语句问题



Python 中循环语句问题。

网络课堂：http://bbs.itzcn.com/thread-12373-1-1.html

现在有这样一段程序：

```
start = 10
delta = 5
for i in range(start, start+delta):
       start = 200
       print start
       print i
       print range(start, start+delta)
```

执行结果如下：

```
>>>
200
10
[200, 201, 202, 203, 204]
200
11
[200, 201, 202, 203, 204]
200
12
[200, 201, 202, 203, 204]
200
13
[200, 201, 202, 203, 204]
200
14
[200, 201, 202, 203, 204]
>>>
```

请问：如何才能使循环的次数随着 start 的变化而变化呢？

【解决办法】你想要的效果恐怕使用 for 循环很难完成，只能使用 while 循环。下面是使用 while 语句的代码。

```
start = 10
delta = 5
while start:
    start+=1
    print start
    print range(start, start+delta)
    if start==17:
        break
```

保存代码后运行程序，执行结果如下：

```
>>>
11
[11, 12, 13, 14, 15]
12
[12, 13, 14, 15, 16]
13
[13, 14, 15, 16, 17]
```

```
14
[14, 15, 16, 17, 18]
15
[15, 16, 17, 18, 19]
16
[16, 17, 18, 19, 20]
17
[17, 18, 19, 20, 21]
>>>
```

这就是想要的效果了。

3.7　习　　题

一、填空题

(1) 有这样一句代码:

```
bool('My name is dcy')
```

程序运行时，得到的结果是__________。

(2) 例如有一个猜拳的游戏，如果你出剪刀，说明你和对方打平手。具体的代码情况如下所示:

```
chu='剪刀'
you=raw_input('你出什么：')
if you==chu:
    print '恭喜您，你们打了平手，请重新进入下一轮'
    _______ you=='石头':
    print '您赢了！'
else:
    print '不好意思，你输了！'
```

在上述代码横线处应填入的关键字是__________。

(3) 在 Python 语言中，我们使用__________语句可以在任何条件为真的情况下循环执行同一个代码块。

(4) 如果想使一个集合或者字典中的每个元素都执行一个代码块，我们可以使用__________循环语句。

(5) 有一个循环语句，如果我们想跳出整个循环，可以使用关键字__________来结束循环。

(6) 在 Python 语言中，如果我们想跳过循环体的某个部分，可以使用关键字__________。

(7) Python 语言中，在多个条件语句中可以作为占位符的关键字是__________。

二、选择题

(1) 下面几个选项中，使用条件语句语法正确的是__________。

A.

```
a=3
b=4
if a==b:
```

```
    print 'a 和 b 的值相等'
elsif a>b:
    print 'a 比 b 的值稍大一些'
elsif a<b:
    print 'a 比 b 的值小一些'
else:
    print 'a 和 b 的值无法比较'
```

B.

```
a=3
b=4
if a==b:
    print 'a 和 b 的值相等'
elseif a>b:
    print 'a 比 b 的值稍大一些'
elseif a<b:
    print 'a 比 b 的值小一些'
else:
    print 'a 和 b 的值无法比较'
```

C.

```
a=3
b=4
if a==b:
    print 'a 和 b 的值相等'
else if a>b:
    print 'a 比 b 的值稍大一些'
else if a<b:
    print 'a 比 b 的值小一些'
else:
    print 'a 和 b 的值无法比较'
```

D.

```
a=3
b=4
if a==b:
    print 'a 和 b 的值相等'
elif a>b:
    print 'a 比 b 的值稍大一些'
elif a<b:
    print 'a 比 b 的值小一些'
else:
    print 'a 和 b 的值无法比较'
```

(2) 在 Python 语言中，我们使用迭代器的__________方法时会出现 StopIteration 异常。

A. readline　　B. iter　　C. next　　D. nextline

(3) 在 Python 语言中，我们使用__________迭代可以将两个序列“压缩”到一起，然后返回一个元祖的列表。

A. open　　B. zip　　C. enumerate　　D. range

(4) 有这样一段代码：

```
items=['dcy','admin','mxl','another','happy','sorry']
i=1
```

```
while i<=1:
    i+=1
    print '---------------循环开始--------------'
    for danqu in items:
        if danqu=='sorry':
            break
        if danqu=='admin':
            continue
        print '第',i-1,'次显示的名称：',danqu
```

运行程序，其执行结果正确的是__________。

A.

```
第 1 显示的名称： dcy
第 1 显示的名称： mxl
第 1 显示的名称： another
第 1 显示的名称： happy
```

B.

```
第 1 显示的名称： happy
第 1 显示的名称： mxl
第 1 显示的名称： another
第 1 显示的名称： dcy
```

C.

```
第 1 显示的名称： dcy
第 1 显示的名称： another
第 1 显示的名称： mxl
第 1 显示的名称： happy
```

D.

```
第 1 显示的名称： dcy
第 1 显示的名称： mxl
第 1 显示的名称： happy
第 1 显示的名称： another
```

三、上机练习

上机练习：输出全体考生的成绩。

通过本章的学习，对 Python 语言中的控制流程语句有了进一步的了解。本章主要强调的是循环语句、跳转语句以及迭代语句的使用，那么本次上机练习就是针对循环语句。

本次上机练习的主要目的是：本班举行了一次内测，想把学生的成绩打印输出，使教师能更好地了解学生对知识的掌握程度，以便对症下药。

其输出结果如下：

```
>>>
apple 的成绩是： 50
zone 的成绩是： 60
hao 的成绩是： 89
admin 的成绩是： 90
thowe 的成绩是： 75
yyyy 的成绩是： 62
jieji 的成绩是： 80
dcy 的成绩是： 95
sorry 的成绩是： 60
dream 的成绩是： 67
happy 的成绩是： 100
>>>
```



第 4 章　可复用的函数和模块

内容摘要

函数是一个能完成特定功能的代码块，可在程序中重复使用，以减少程序的代码量和提高程序的执行效率。模块可把一个复杂的程序按功能分开，分别存放到不同的文件中，使程序更容易维护和管理。

本章将介绍 Python 中模块和函数的概念。结构化程序设计可以把复杂的问题分解为若干个子任务，然后针对子任务定义实现的模块和函数。本章将详细介绍 Python 模块和函数的创建与使用方法。

学习目标

- 了解 Python 程序的结构设计。
- 掌握模块的创建。
- 掌握导入模块的方法。
- 掌握模块的内置函数。
- 熟练定义包。
- 掌握定义函数的步骤。
- 掌握如何调用函数。

4.1 Python程序的结构

Python 程序由包(package)、模块(module)和函数(function)组成，本节将介绍 Python 中这三者之间的关系。

 **视频教学：** 光盘/videos/04/ Python 程序的结构.avi　　长度：4 分钟

Python 程序其实是由包、模块和函数三者组成的。其中，包是由一系列模块组成的集合。模块是处理某一类问题的函数和类的集合。三者之间的关系如图 4-1 所示。

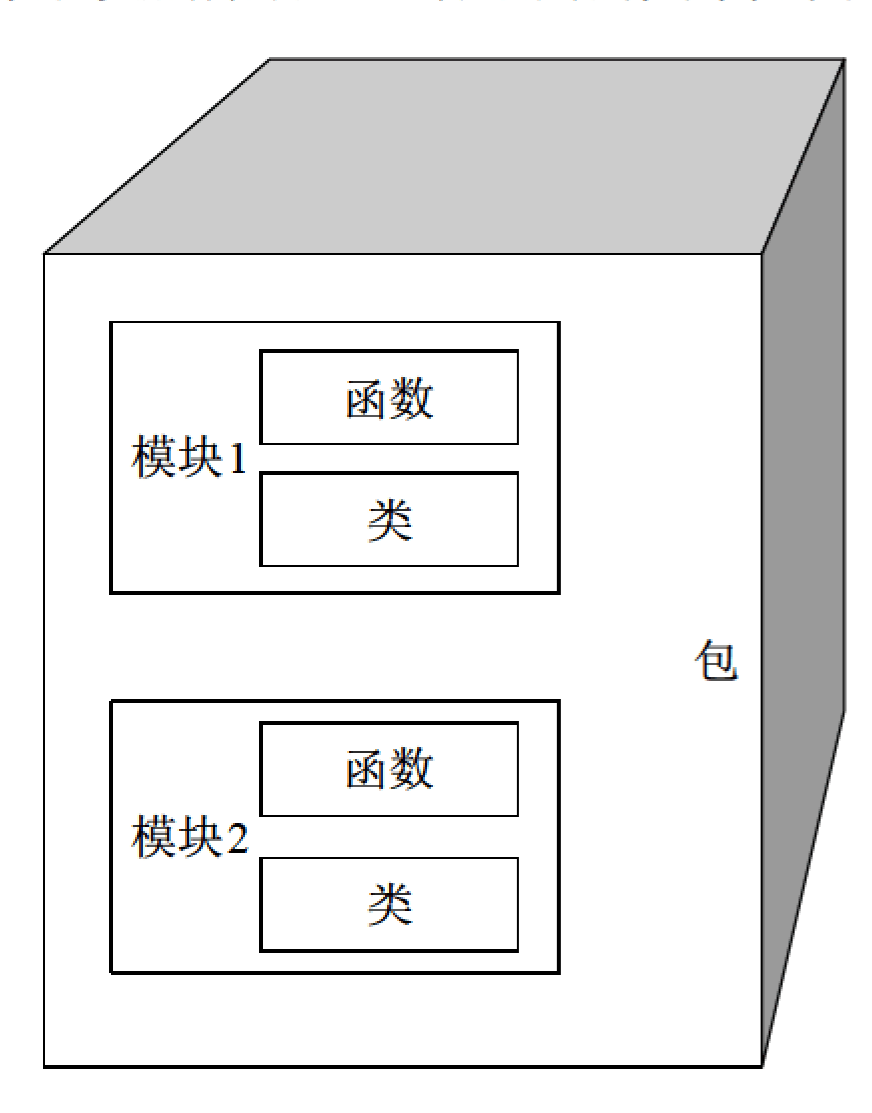


图 4-1　包、模块和函数之间的关系

图 4-1 中的函数和类表示 0 个或多个。包其实就是一个完成特定任务的工具箱，Python 提供了许多有用的工具包，如字符串处理、图像用户接口、Web 应用、图形图像处理等。使用这些工具包，可以提高程序员的程序开发效率，减少编程的复杂度，达到代码重用的效果。



Python 提供的许多工具包和模块安装在 Python 的安装目录下的 Lib 子目录中。

例如，在 Python 安装目录下的 Lib 目录下，有一个 sqlite3 文件夹，该文件夹就是一个包，这个包用于完成连接 sqlite3 数据库的基本操作。在 sqlite3 文件夹下有一个 test 的子包，还有一个__init__.py 文件，该文件是 sqlite3 包的注册文件，如果没有该文件，Python 将不能识别 sqlite3 包。



包必须至少包含一个__init__.py 文件，该文件的内容可以为空。__init__.py 用于标识当前文件夹是一个包。

4.2　计算相对年龄

函数是可重用的程序段，即可重复多次调用的代码段。可以通过输入参数值，返回需要的结果。本节将讲解如何定义函数。

 **视频教学：** 光盘/videos/04/函数的定义.avi　　长度：4 分钟

4.2.1　基础知识——函数的定义

函数的定义非常简单，使用保留字 def 声明即可。在定义函数的时候，需要定义该函数返回值的类型。Python 函数的语法如下：

```
def function_name(arg1, arg2 [, ...]):
    statement
    [return value]
```

其中，返回值不是必须的，如果没有 return 语句，则 Python 默认返回值为 None。函数通过 def 保留字定义，def 保留字后是函数的标识符名称，然后是一对圆括号。函数的参数放在圆括号中，参数的个数可以是一个或多个，参数之间用逗号隔开，这种参数称为形式参数。Python 支持形式参数的默认值语法，默认值的赋值是可选的。函数名称及参数定义之后，用冒号作为结束标识。冒号下面就是函数的主体部分。下面来定义一个函数。

```
# 函数的定义
def login (username , password):
    if(username == 'admin') and (password == 'admin'):
        print "登录成功! "
    else:
        print "登录失败"
```

在该段代码中，定义了一个名称为 login()的函数，该函数有两个参数，分别为 username 和 password。在函数的主体部分使用了 if 条件控制语句，判断参数的值是否合法，如果合法，则输出“登录成功”的信息；如果失败，则输出“登录失败”的信息。

函数已经创建成功，那么怎么调用该函数呢？函数可以在当前的文件中调用，也可以在其他模块中调用。函数调用的格式如下：

```
function_name(arg1, arg2 [, ...])
```

函数的调用采用函数名加一对圆括号的方式，圆括号内的参数是传递给函数的具体值。函数调用中的实参列表分别与函数定义中的形参列表对应。图 4-2 说明了函数实际参数和形式参数的对应关系。

下面调用已经创建好的参数 login()，代码如下：

```
login('admin','admin')
```

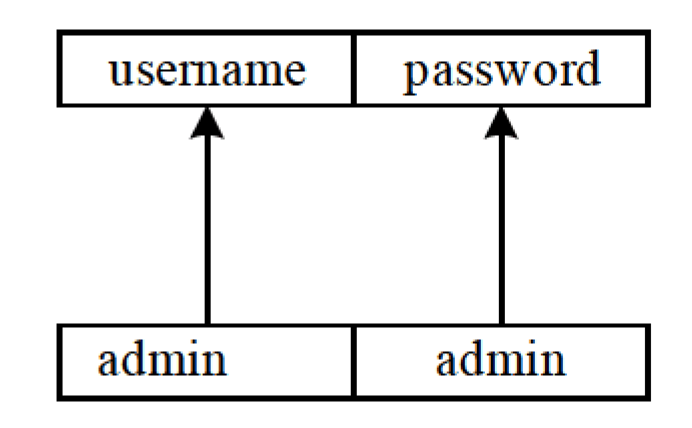


图 4-2　实际参数与形式参数的对应关系

因为在 login()函数中已经使用了 print 语句针对不同的情况输出不同的提示信息，因此在调用 login()函数时，不需要再次使用 print 语句，直接调用即可。保存并运行，输出结果如下：

```
登录成功！
```



实际参数必须与形式参数一一对应，参数的顺序和参数的类型都需要一一对应，否则将出现错误。如果参数提供默认值，顺序可以不一致。

4.2.2　实例描述

小明今年 5 岁了，刚上学前班，孩子既聪明又可爱，在学校里经常受到老师的表扬。为了培养孩子的思考能力，有一天父亲对小明说："爸爸今年 26 岁了，那么当你 7 岁的时候爸爸多大呢？"下面使用 Python 中的自定义函数，帮小明算一下他父亲的年龄吧！

4.2.3　实例应用

【例 4-1】计算父亲的年龄。

(1) 创建一个 Python 文件，命名为 countAge.py。

(2) 编辑 countAge.py 文件，在其中定义函数 countAge()，此函数将返回小明 7 岁时父亲的年龄，代码如下：

```
# 定义计算年龄的函数
def countAge(yson,setAge,yfather):
# 求年龄差
  differAge=(int)(setAge-yson)
  #计算父亲的实际年龄
  realFatherAge=(int)(yfather+differAge)
  return realFatherAge
  #调用自定义函数
```

(3) 调用 countAge.py 文件中的 countAge()函数：

```
print countAge(5,7,26)
```

4.2.4　运行结果

运行程序，输出结果如下：

```
---------- python ----------
28
```

4.2.5　实例分析



源码解析

上述例子主要讲述了如何创建自定函数 countAge()以及怎样调用它的方法。用 def 保留字定义函数 countAge()，并设置适当的参数，然后根据条件计算出小明父亲的年龄，再用 print 语句调用自定义函数 countAge()，最后将该函数返回的最终结果输出。

4.3　验证用户注册信息

函数取得的参数需要用户提供，这样函数就可以利用这些参数做一些事情，完成一项功能。这些参数和 Python 中的变量一样，只不过参数的值是在调用函数时定义的，而不是在函数内部定义和赋值的。而对于一些函数，有些参数是可选的，如果用户不需要为这些参数提供值，那么这些参数就使用它的默认值。本节将详细介绍函数中的参数传递与默认参数值。



视频教学： 光盘/videos/04/函数形参与默认参数值.avi　　长度：11 分钟

4.3.1　基础知识——函数形参与默认参数值

参数的传递有两种方式：值传递和引用传递。参数在函数定义的圆括号内指定，用逗号隔开，当调用函数的时候，也需要以同样的方式提供值(函数中的参数名称为形参，用户提供给函数调用的值称为实参)。Python 通过名字绑定的机制，把实际参数的值和形式参数的名称绑定在一起，即把形式参数传递到函数所在的局部命名空间中，形式参数和实际参数指向内存中的同一个存储空间。

1. 默认参数值

函数的参数支持默认值。当某个参数没有传递实际值时，函数将使用默认参数计算。例如，可以向 login()函数的 username 参数和 password 参数分别提供一个默认值。

```
# 函数的定义
def login (username = "maxianglin" , password = "maxianglin"):
    if(username == 'admin') and (password == 'admin'):
        print "登录成功！"
    else:
        print "登录失败"
```

在该段代码中，使用赋值表达式的方式定义了 username 参数的默认值为 maxianglin，password 参数的默认值为 maxianglin。下面来调用 login()函数，请具体查看一下提供不同的实

际参数运行的结果有何不同。

```
login('admin','admin')
login('admin')
login(password='admin')
login()
```

第 1 行代码中提供了两个实际参数，即 username 形式参数的实际值为 admin，将默认的 maxianglin 值覆盖；password 形式参数的实际值为 admin，覆盖了原有的默认值，输出结果如下：

```
登录成功!
```

第 2 行代码中提供了一个实际参数，即 username 形式参数的实际值为 admin，覆盖原有的默认值，而 password 形式参数没有提供实际值，因此使用默认值，即 maxianglin。可见用户名和密码不是合法的，提示“登录失败”的信息，输出结果如下：

```
登录失败
```

第 3 行代码中也提供了一个实际参数，即 password 形式参数的实际值为 admin，而 username 的实际值保持了原有的默认值 maxianglin，因此该用户提供的用户名和密码也是不合法的，输出结果如下：

```
登录失败
```

第 4 行代码中没有提供任何实际参数，即参数 username 和 password 都采用默认值 maxianglin，因此该用户提供的用户名和密码都是不合法的，输出结果如下：

```
登录失败
```

2. 列表参数值

参数可以是变量，也可以是元组、列表等内置数据结构。例如：

```
# 函数的定义
def login (usernames = [] , password = "admin"):
    username = usernames[0]
    if(username == 'admin') and (password == 'admin'):
        print "登录成功! "
    else:
        print "登录失败"
```

在定义函数 login()时，同时定义了两个形式参数：一个是名称为 usernames 的列表，另一个是名称为 password 的变量，并赋予默认值 admin。在该函数的主体部分，首先声明了一个名称为 username 的变量，并获取参数 usernames 列表中的第一个元素作为值。接着使用 if 条件控制语句来判断该用户是否是合法用户。之后调用 login()函数，将列表['admin','maxianglin']作为实际参数传递给 login()函数，password 参数使用默认值。

```
login(['admin','maxianglin'])
```

运行代码，输出结果如下：

```
登录成功!
```

3. 可变长度参数值

在程序开发过程中，常常需要传递可变长度的参数。在函数的参数前使用标识符*，可以实现这个要求。*可以引用元组，将多个参数组合在一个元组中。

```
# 传递可变参数
def login (* userpwds):
    username = userpwds[0]
    password = userpwds[1]
    if(username == 'admin') and (password == 'admin'):
        print "登录成功！"
    else:
        print "登录失败！"
```

在该段代码中，首先在 login()函数中使用标识符*定义了一个可变长度的参数 userpwds。接着在函数的主体部分声明变量 username，并获取 userpwds 元组中的第 1 个值作为 username 变量的值，然后声明了 password 变量，并获取 userpwds 元组中的第 2 个值作为 password 变量的值。最后判断 username 和 password 的值，检测用户是不是合法用户。下面来调用 login()函数，传入不同的参数以查看该函数的不同结果。

```
login('admin','admin')
login('maxianglin','maxianglin')
```

在第 1 行代码中，由于参数使用了*标识符，因此传入的实际参数被“封装”到一个元组中。其中的两个参数 admin 成为 userpwds 元组的元素。输出结果如下：

```
登录成功！
```

第 2 行代码中，使用同样的方式调用了 login()函数，传入的参数不同，被“封装”后的 userpwds 元组的值也不同。在函数 login()主体内部的 username 和 password 参数的值也不同，最后输出的结果也不同。

```
登录失败！
```

4. 字典类型参数值

Python 还提供了另一个标识符**，在形式参数前面添加**，可以引用一个字典作为参数。根据实际参数的赋值表达式生成字典。例如，下面这段代码实现了在一个字典中匹配元组的元素。

```
# 传递字典类型的参数
def login (**userpwds):
    keys=userpwds.keys()
    username=''
    password=''
    for key in keys :
        if 'username'==key:
            username=userpwds[key]
        if 'password'==key:
            password=userpwds[key]
    if(username == 'admin') and (password == 'admin'):
        print '登录成功！'
    else:
        print '登录失败！'
```

在该段代码中，定义了 login()函数的参数为字典类型。接着在函数的主体部分获取了字典参数的所有 key，然后使用 for…in 循环遍历字典参数的所有 key，判断 key 是否为 username 或者 password，如果成立，则获取相对应的值，并赋值给变量 username 和 password。最后在函数的主体部分判断变量 username 和 password 的值是否合法，即 username 变量的值为 admin，password 变量的值也为 admin。如果合法，则提示用户“登录成功！”，否则提示用户“登录失败！”。下面来调用 login()函数，并传递一个字典类型的实际参数。

```
login(username='admin',password='admin')
```

login()函数的形式参数** userpwds 与实际参数 username='admin'、password='admin'对应，生成一个结构为{username : 'admin', password: 'admin'}的字典。运行该段代码，输出结果如下：

```
登录成功!
```



如果函数的参数类型既有元组(形式参数前加*)，又有字典(形式参数前加**)，那么*必须写在**的前面，这是语法规定。

4.3.2 实例描述

互联网的发展，为我们的工作、学习和生活带来了很大方便。相信你对注册账号并不陌生。小李最近刚学了 Python 这门课程，他想运用自己所学的知识来实现用户注册的功能。

具体功能及要求如下。

(1) 对用户输入的用户名称进行验证，要求用户名称的长度在 3～20 之间。

(2) 对用户输入的账号进行验证，要求输入的账号必须为数字。

(3) 对用户输入的电话号码进行验证，要求电话号码必须为数字且长度要在 7～11 之间。

下面利用 Python 中的函数形参与默认值，帮“小李”实现这些功能吧。

4.3.3 实例应用

【例 4-2】验证用户注册信息。

(1) 创建一个 Python 文件，命名为 userRegister.py。

(2) 编辑用户注册账号所使用的函数 registerUser()，在该函数中设置参数的默认值：userName="py"，idcard="python"，tel="123"。具体实现用户注册功能的代码如下：

```
#创建函数并设置其默认值
def registerUser(userName="py",idcard="python",tel="123"):
    #提示输入用户名称
    userName=raw_input("请输入用户名称:")
    #判断用户输入名称的长度
    if(len(userName)>3 and len(userName)<20):
        idcard=raw_input("请输入您的账号:")
        #判断用户输入的账号是否为数字
        if(idcard.isdigit()):
            tel=raw_input("请输入您的电话号码:")
            #判断电话号码的长度及格式
```

```
            if(tel.isdigit() and len(tel)>7 and len(tel)<11):
                print "恭喜您，注册成功！"
            else:
                print "电话格式或长度不正确！"

        else:
                    print "账号格式不正确，请输入数字！"
    else:
        print "用户名称的长度必须要在 3-20 之间！"
```

(3) 调用函数 registerUser()，代码如下：

```
print registerUser('python','3232','1111134344')
```

4.3.4　运行结果

运行 userRegister.py 文件，Python 解释器输出“请输入用户名称：”等待用户输入。当用户按照注册要求依次正确输入时，结果如图 4-3 所示。如果用户输入的用户名称不合要求时，结果如图 4-4 所示。当用户输入的账号不合要求时，结果如图 4-5 所示。当用户输入的电话号码不合要求时，结果如图 4-6 所示。

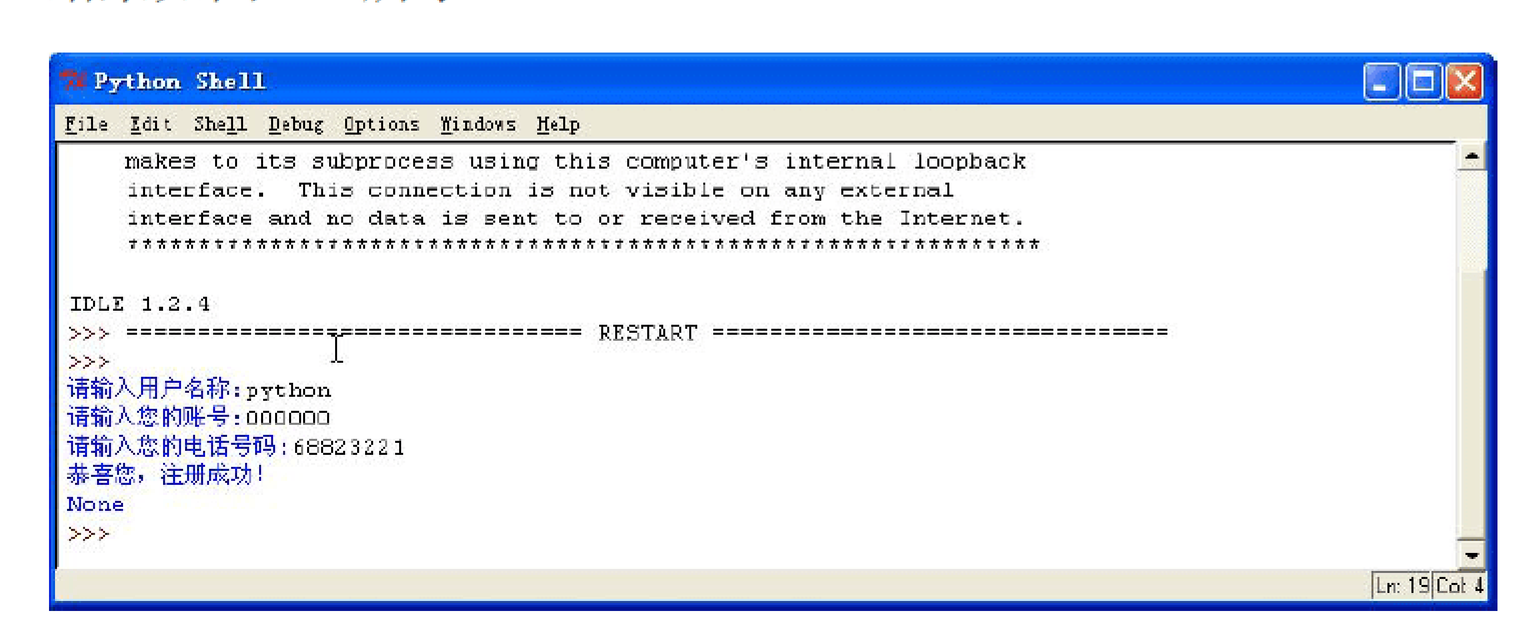


图 4-3　按要求正确输入结果

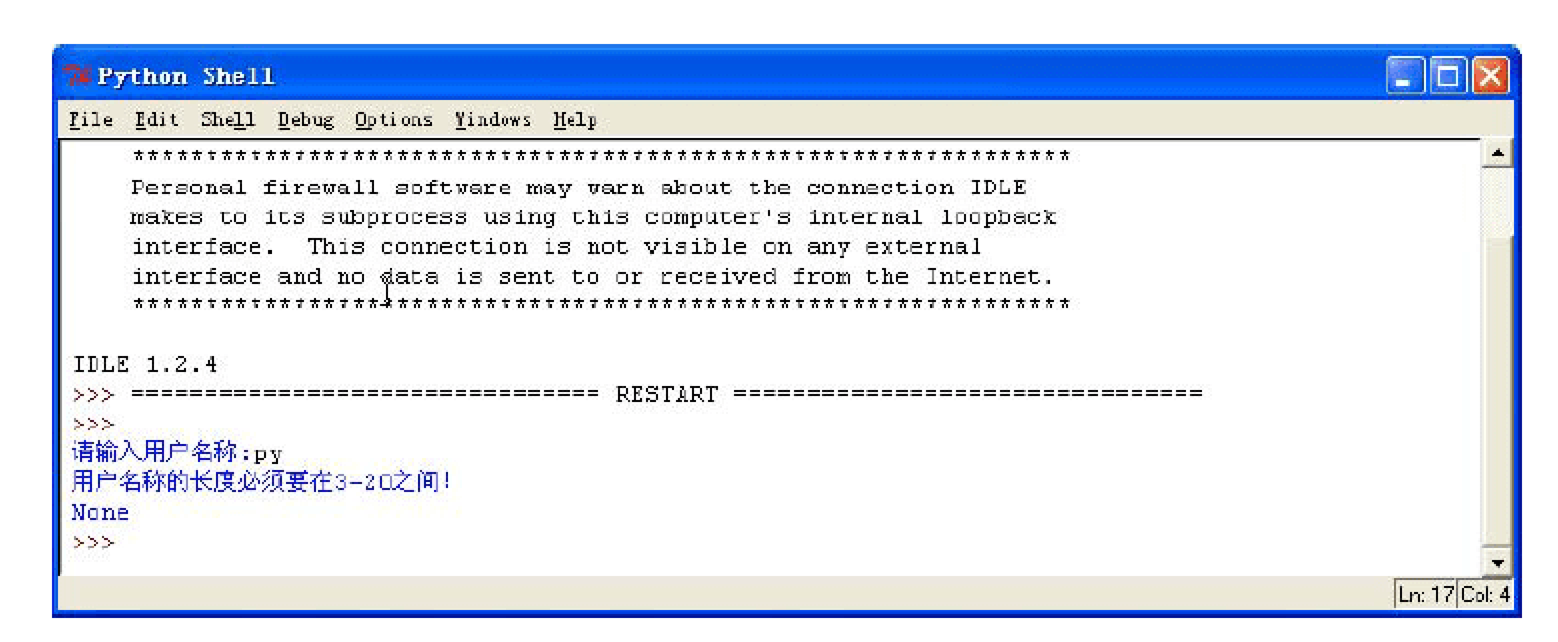


图 4-4　用户名不合要求

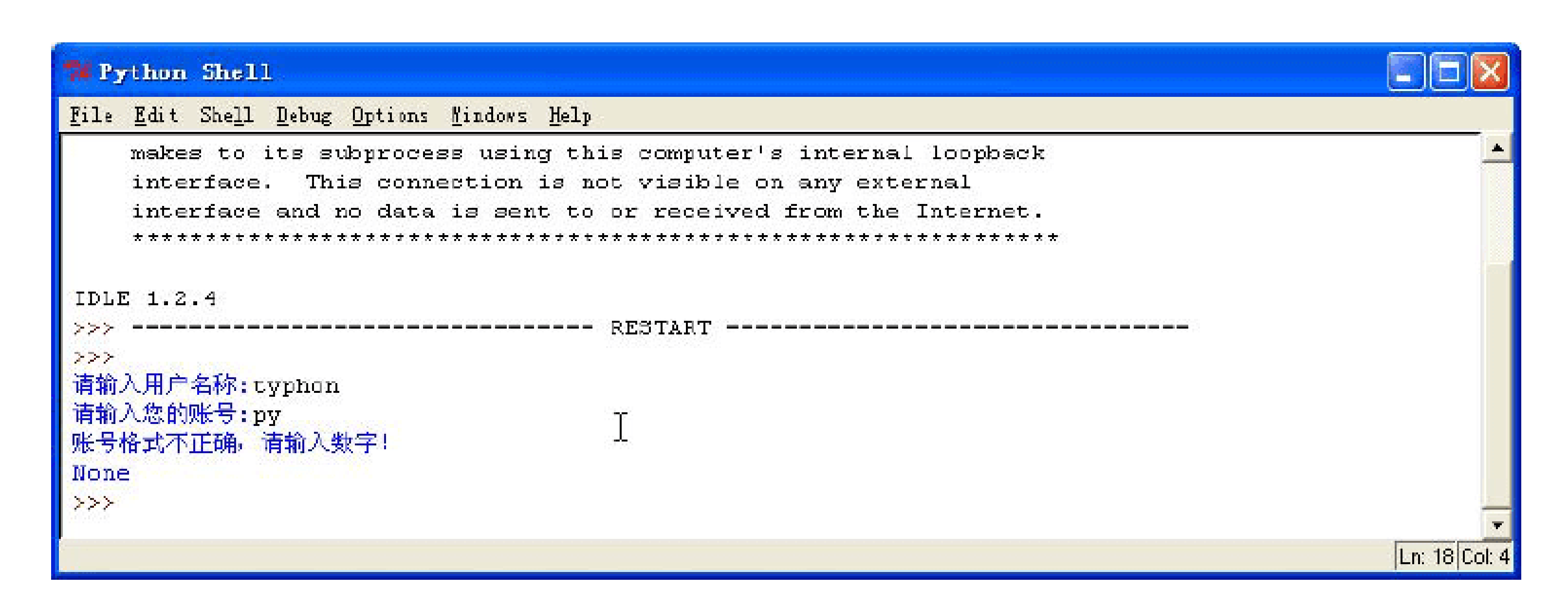


图 4-5　账号不合要求

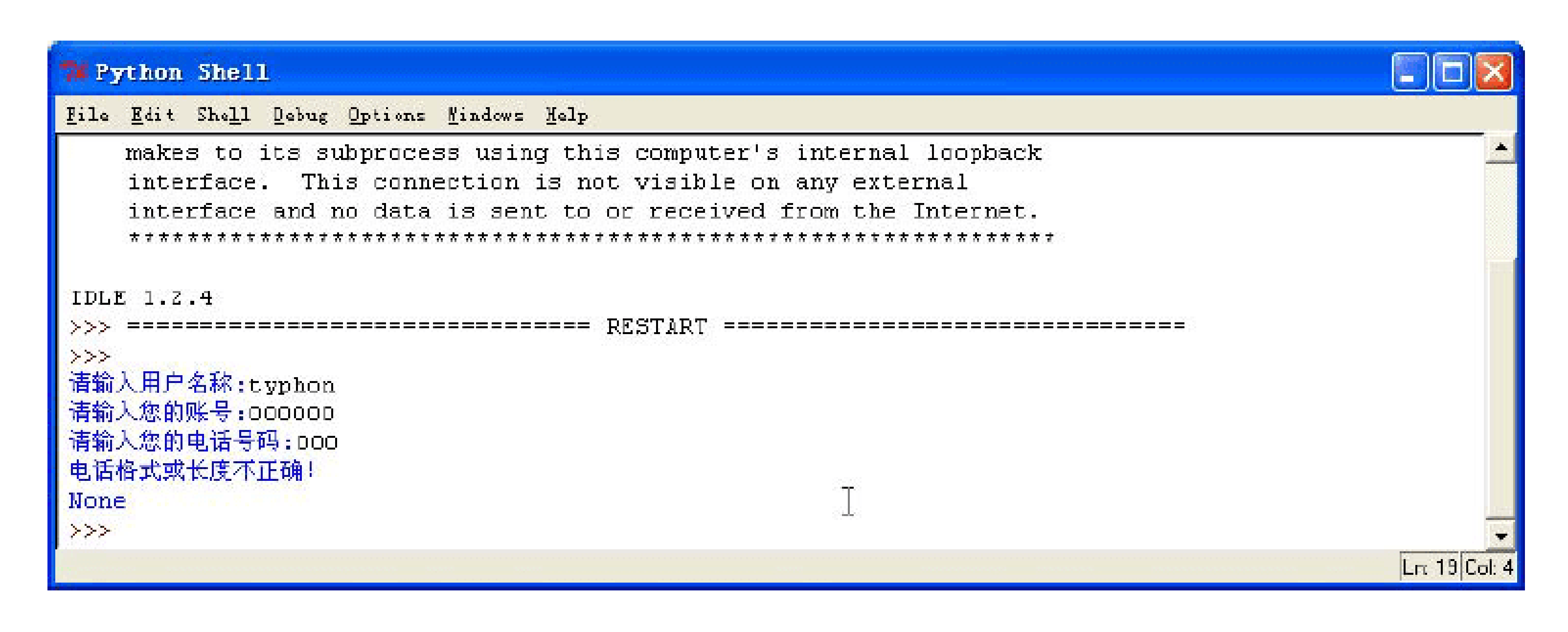


图 4-6　电话号码不合要求

4.3.5　实例分析


源码解析

在上述实例中，虽然在函数 registerUser()中分别设置了默认值 userName="py"，idcard="python"，tel="123"，但当按照提示重新输入实际参数值时，形式参数的实际值将覆盖原有的默认值。在函数 registerUser()中，按照实际输入的参数值并根据条件做出判断，运行得到不同的结果。

4.4　判断是否闰年

如果一个函数需要返回结果，则需要使用 return 语句。return 后的语句可以是一个变量，也可以是一个格式正确的表达式。本节将详细介绍含有返回值的函数的定义和调用。


视频教学：光盘/videos/04/函数的返回值.avi　　　　**长度：5 分钟**

4.4.1　基础知识——函数的返回值

在程序的开发中，往往需要通过函数来获取一个值。这是如何实现的呢？其实方法很简单，

只需要在定义函数时，在函数的最后使用 return 语句来将需要得到的值返回即可。下面定义一个含有返回值的函数 login()，并为该函数添加 return 语句，使其更加完善。

```
# 定义含有返回值的函数
def login (username,password):
    msg=''
    if(username == 'admin') and (password == 'admin'):
        msg="登录成功！"
    else:
        msg='登录失败！'
    return msg
```

在该函数的主体部分，首先声明了一个名称为 msg 的变量，用来获取该函数将要返回的值。接着使用 if 条件控制语句来判断用户名和密码是否合法，如果合法，则 msg 的值为“登录成功！”，否则为“登录失败！”。最后使用 return 语句将获取的结果返回，这里 return 后是一个变量。下面来调用 login()函数。

```
print login('admin', 'admin')
```

细心的同学应该会发现，这里的调用与前面章节中的调用有所不同。在前面的章节中，并没有使用 print 语句来调用 login()函数，这里却需要使用 print 语句。这是因为，前面章节中定义的 login()函数，已经使用了 print 函数将最后的结果输出了，而这里的 login()函数只是使用了 return 语句将最后的结果返回，并没有输出，因此在调用时需要使用 print 语句将最后的结果输出。运行该段代码，输出结果如下：

```
登录成功!
```

如果需要返回多个值，可以将需要返回的值全部“打包”到元组中。在调用时，对返回的元组“解包”即可。下面再定义一个函数，实现自增 5 并返回结果，代码如下：

```
# 定义返回多个值的函数
def operat (x, y, z):
    x = x + 5
    y = y + 5
    z = z + 5
    oper = [x, y, z]
    numbers = tuple(oper)
    return numbers
```

在该段代码中定义了一个 operat()函数，该函数有 3 个参数，分别将传递过来的 3 个参数自增 5 之后，使用 oper = [x , y , z]将这 3 个参数“打包”到一个名称为 oper 的列表中，然后使用 tuple()函数将列表装到元组中，最后将元组返回。下面调用 operat()函数获得返回的元组，并“解包”到 3 个变量中，最后将这 3 个变量输出。

```
x,y,z = operat(1,2,3)
print x,y,z
```

运行上段代码，输出结果如下：

```
6 7 8
```

4.4.2 实例描述

工作了一天，老王拖着疲惫的身体回到家中，刚刚推门进入客厅还没来得及休息，儿子从书房跑了出来，对父亲说："爸爸，老师今天给我们留了这样一道问题，让我们算一算今年是平年还是闰年？我算了好久，现在还没有算出结果，你能帮我算一下吗？"

下面利用 Python 中函数的返回值，帮老王算一下吧！辛苦了一天，让他多休息一会。

4.4.3 实例应用

【例 4-3】判断今年是否闰年。

(1) 创建一个 Python 文件，命名为 judgeYear.py。

(2) 编辑用于判断平年与闰年的函数 judgeYear()，在该函数中声明一个名称为 message 的变量，用于返回函数的值。代码如下：

```
#定义用于判断瑞年与平年的函数
def judgeYear(year):
    #声明变量用于返回值
    message=''
    #判断是闰年与平年的条件
    if((year%4 ==0)and(year%100!=0)or(year%400==0)):
        message="今年是闰年！"
    else:
        message="今年是平年！"
    return message
```

(3) 调用函数 judgeYear()，并传入参数 2011，代码如下：

```
print judgeYear(2011)
```

4.4.4 运行结果

当运行程序时，根据年份 2011 判断出今年是平年还是闰年，结果如下：

```
---------- python ----------
今年是平年!
```

4.4.5 实例分析



源码解析

在本实例中，主要讲述了函数 judgeYear()如何通过 return 语句把所需的值返回，使用 if 条件控制语句来判断今年是闰年还是平年，如果满足条件(year%4 ==0) and (year%100!=0) or (year%400==0)，那么今年就是闰年，否则为平年。

4.5　调用模块函数添加用户

Python 程序是由一个个模块组成的，模块是 Python 的一个重要概念。其实，一个 Python 文件就是一个模块。下面来介绍模块的创建方法。

 视频教学：光盘/videos/04/模块的创建.avi　　长度：5 分钟

4.5.1　基础知识——模块的创建

模块是把一组相关的函数或代码组织到一个文件中，即一个文件就是一个模块。模块是由代码、类和函数组成的，其中类和函数可以有 0 个或者多个。例如，创建一个名称为 myFirstModule.py 的文件，即定义了一个名称为 myFirstModule 的模块。在该模块中定义两个函数和一个类，并在该类中定义一个方法，代码如下：

```
# 自定义模块
def myFun1 ():
    print 'myFirstModule myFun1()'

def myFun2 ():
    print 'myFirstModule myFun2()'

class MyClass:
    def myClassFun (self):
        print 'myFirstModule MyClass myClassFun()'
```

在 myFirstModule 模块中定义了两个函数，分别为 myFun1()和 myFun2()。同时定义了一个名称为 MyClass 的类，在该类中定义了一个名称为 myClassFun()的方法。

接着在 myFirstModule.py 文件所在的目录下创建一个名称为 call_myFirstModule.py 的文件，并在该文件中调用 myFirstModule 模块中的函数和类，编辑代码如下：

```
# 调用自定义模块的类和函数
import myFirstModule

myFirstModule.myFun1()
myFirstModule.myFun2()
myclass=myFirstModule.MyClass()
myclass.myClassFun()
```

在 call_myFirstModule.py 文件中，首先使用 import 关键字导入 myFirstModule 模块，接着使用 myFirstModule 作为前缀分别调用 myFirstModule 模块中的两个函数，然后创建了 myFirstModule 模块中 MyClass 类的实例 myclass，这里也需要使用 myFirstModule 作为前缀来调用类。最后调用类中的方法 myClassFun()。运行该段代码，输出结果如下：

```
myFirstModule myFun1()
myFirstModule myFun2()
myFirstModule MyClass myClassFun()
```



myFirstModule.py 和 call_myFirstModule.py 文件必须放在同一目录下。这里放在了项目的根目录下，或者放在 sys.path 所列出的根目录下，否则 Python 解释器找不到自定义模块的位置，从而无法调用模块中的类或函数。

在编辑 Python 程序时，当使用 import 关键字导入一个模块时，系统首先会查找当前路径，然后查找 lib 目录、site-packages 目录(Python/Lib/site-packages 目录)和环境变量 PYTHONPATH 设置的目录。

4.5.2 实例描述

闲暇之余，小王在网上看到一篇不错的文章，想参与评论一番。当他把自己所要评论的信息准备提交时，系统提示他不属于本网站的会员，无权参与文章评论，希望他注册成为本网站的会员。注册会员当然少不了用户名称，试想一下：在 Python 中如何利用模块的创建来提示用户输入用户名称？

4.5.3 实例应用

【例 4-4】调用模块中的函数来添加用户。

(1) 创建一个 Python 文件，命名为 addPerson.py。

(2) 编辑注册用户所使用的 addPerson 模块，在该模块中定义一个名称为 Person 的类，并在该类中定义一个 addPer()方法，该方法实现了添加用户的功能。代码如下：

```
class Person:
    def addPer (person):
        #提示用户输入用户名称
        userName=raw_input("please input you name:")
        #打印输出所输入的用户名称
        print userName
        #判断用户所输入的名称是否为空
        if userName!="":
            print "person add success"    #如果用户名不为空提示添加成功
        else:
            print "person add failure"    #如果用户名称为空提示添加失败
```

(3) 再次创建一个 Python 文件，命名为 showperson.py。

(4) 在 showperson.py 文件中首先导入 addPerson 模块，然后在该模块的 Pseron 类中调用添加用户的方法。代码如下：

```
#引入所创建的模块
import addPerson
person=addPerson.Person()        #调用模块中的类
person.addPer()                  #执行类中的添加方法
```

4.5.4　运行结果

运行 showperson.py 文件，Python 解释器输出“please input you name:”，等待用户输入。当用户输入用户名之后，按回车键提示用户“person add sucess”信息，如图 4-7 所示；否则提示“person add failure”信息，如图 4-8 所示。

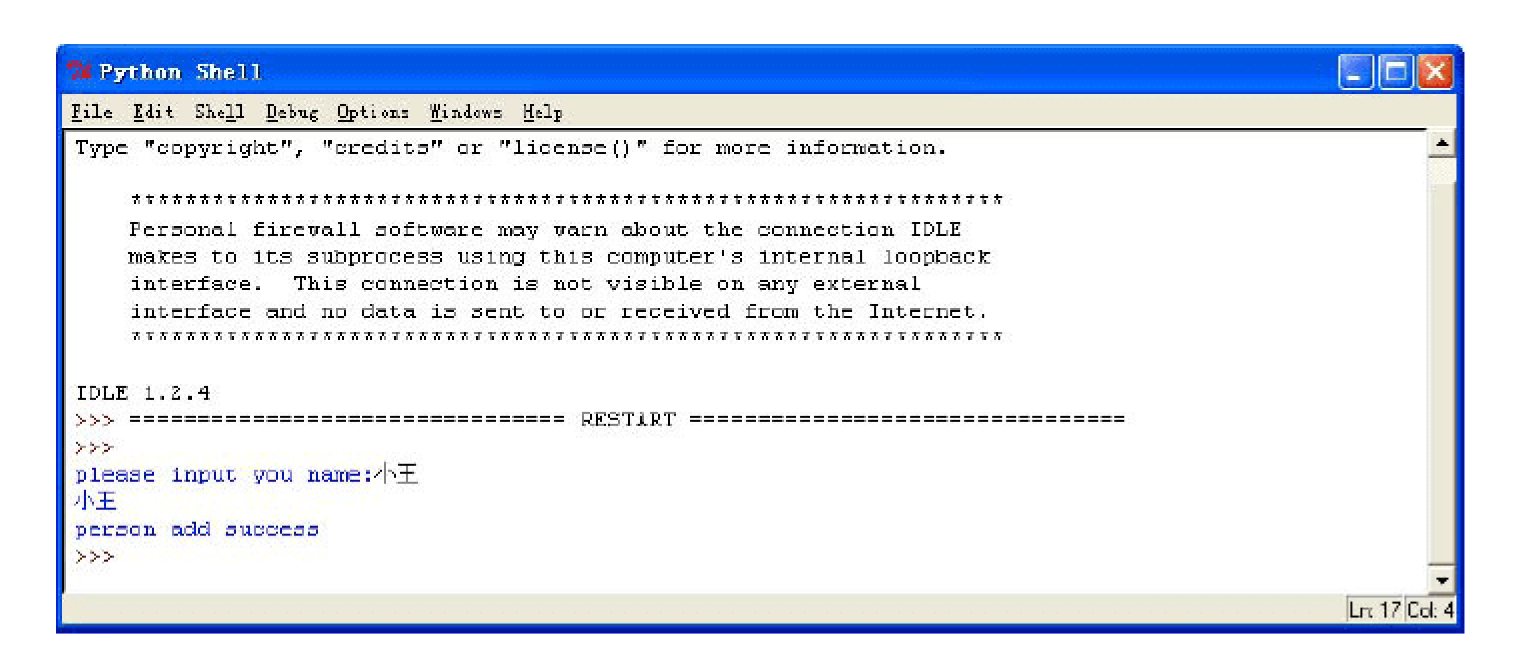


图 4-7　输入用户名称的结果

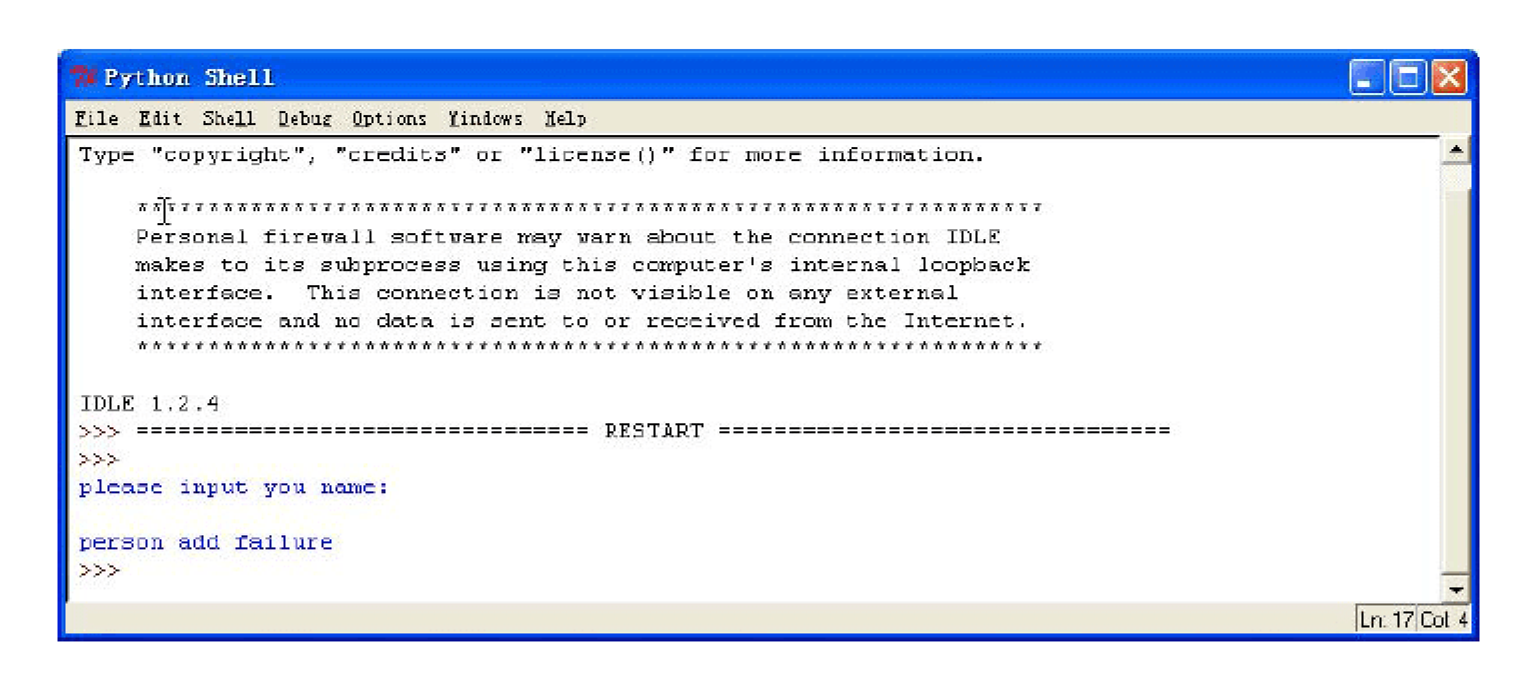


图 4-8　不输入用户名称的结果

4.5.5　实例分析

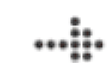


源码解析

在上述案例中，在 addPerson 模块中自定义了添加用户类 Person，其中声明了用户输入的提示信息，而且还判断了不同的操作所运行的结果不同，为其调用做准备。在 showperson.py 文件中引入了 addPerson.py 模块，在这个过程中调用了其添加用户的方法 addPer()，最后输出结果。

4.6　重新设置安全密码

在使用模块中的函数或类之前，首先需要导入该模块，在上一节的案例中使用了模块的导入，本节将详细介绍模块的导入方法。



 视频教学：光盘/videos/04/模块的导入.avi　　长度：6 分钟

4.6.1 基础知识——模块的导入

正如前面章节中使用过的一样，模块的导入需要使用 import 语句，模块导入的格式如下：

```
import module_name
```

该条语句可以直接导入一个模块，调用模块已经定义的类或函数，需要以模块名作为前缀。调用模块中的函数的语法格式如下：

```
module_name.function_name
```

在调用模块中的类时，需要创建类的对象，然后再通过类对象调用类中的方法。语法格式如下：

```
class_object=module_name.class_name
class_object.class_fun_name
```

当然采用这种方式比较麻烦，每次调用模块中的函数或类时，都需要使用模块名作为前缀来调用。例如，在一个文件中需要多次用到模块中已经定义过的方法或类时，是否每次都需要使用模块名作为前缀来调用呢？为了解决这个问题，Python 中的 from …import 语句可以将模块中的类或函数导入，从而不需要使用模块名作为前缀，from …import 的语法格式如下：

```
from module_name import function_name
```



虽然 from …import 的语句为程序员减少了代码的编写，但是使用 from …import 语句却容易造成代码的可读性差等问题。

导入一个模块下的所有类和函数，可以使用如下语句：

```
from module_name import *
```

此外，同一模块文件支持多条 import 语句。例如，定义一个名为 mySecondModule 的模块，该模块定义一个全局变量 num 和一个函数 myFun()，每次调用 myFun()函数时，都使 num 的值增 1。代码如下：

```
# 定义 mySecondModule 模块，每次调用 myFun()函数，全局变量 num 子增 1
num = 5
def myFun ():
    global num
    num = num + 1
    return num
```

接着创建 call_mySecondModule.py 文件，并在该文件中导入 mySecondModule 模块，然后调用模块中的 myFun()函数，查看全局变量 num 的值。代码如下：

```
# 调用模块中的 myFun()函数，使全局变量 num 的值增 1
import mySecondModule
num=mySecondModule.myFun()
print "调用模块函数，num="+str(num)
```

在该段代码中，首先导入 mySecondModule 模块，然后使用模块名作为前缀调用

mySecondModule 模块中的 myFun()函数，并将函数返回的结果赋值给 num 变量，最后使用 print 语句输出结果。运行程序，输出结果如下：

```
调用模块函数，num=6
```

改变 mySecondModule 模块中全局变量 num 的值，再次调用函数，查看运行结果会发生什么样的变化。接着在 class_mySecondModule.py 文件中编辑如下代码：

```
mySecondModule.num = 10
num=mySecondModule.num
print "改变模块局部变量 num 的值，此时 num="+str(num)
senum=mySecondModule.myFun()
print "当 num 的值为 10 时，调用模块函数，num="+str(senum)
```

在该段代码中，首先将 mySecondModule 模块中的全局变量 num 的值赋值为 10，然后将赋值后的值输出，接着调用 mySecondModule 模块中的 myFun()函数，检测当变量 num 的值发生改变之后，在函数中的调用是否也将发生变化。运行程序，输出结果如下：

```
改变模块局部变量 num 的值，此时 num=10
当 num 的值为 10 时，调用模块函数，num=11
```



Python 中的 import 语句比 Java 中的 import 语句更加灵活。Python 中的 import 语句可以放置于程序中的任意位置，甚至可以存放在条件语句中。大家简单写一个程序，然后把 import 语句放置于程序中的任意位置，检测是否会出错。请大家试一试吧！

4.6.2　实例描述

当我们的账号受到威胁时，或许会选择修改自己的账号密码来加强账号的安全系数。相信大家对修改密码并不陌生，下面就来做一个重置密码的案例。具体要求如下：

首先，当用户修改密码时，必须输入原始密码，如果所输入的原始密码不正确，则提示用户“原始密码输入错误，请重新输入！”，否则提示用户输入新的密码。当用户输入新密码时，要对密码长度进行验证，要求新密码的长度要在 6～18 位之间。当新密码输入完成，提示用户输入确认密码，要求确认密码和新密码必须保持一致，如果两次输入的密码不相同，提示用户“两次输入的密码不一致，修改失败！”，否则提示用户“恭喜您！密码修改成功！”

下面利用 Python 中导入模块的知识来实现上述功能。

4.6.3　实例应用

【例 4-5】加强账号的安全系数，重新设置用户密码。

(1) 创建一个 Python 文件，命名为 updatePwd.py。

(2) 编辑修改账号密码所使用的模块 updatePwd，在该模块中定义一个名称为 updatePassword()的函数，该函数用于实现修改账号密码的功能。代码如下：

```
# -*- coding: UTF-8 -*-
def updatePassword():
    message=""
    oldPwd=raw_input(u"请输入原始密码：")
    if(oldPwd != "python"):
        message="原始密码输入错误，请重新输入！"
    else:
        newPwd=raw input(u"请输入新密码：")
        if(len(newPwd)>6) and (len(newPwd)<18):
            renewPwd=raw_input(u"请输入确认密码：")
            if(renewPwd!=newPwd):
                message="两次输入的密码不一致，修改失败！"
            else:
                message="恭喜您！密码修改成功！"
        else:
            message="密码长度必须在 6-18 位之间"
    return message
```

(3) 再次创建一个 Python 文件，命名为 call_updatePwd.py，用于调用 updatePwd 模块中的函数。

(4) 在 call_updatePwd.py 文件中，首先导入所创建的 updatePwd 模块，然后调用该模块中的函数 updatePassword()。代码如下：

```
# -*- coding: UTF-8 -*-
import sys
import updatePwd
type = sys.getfilesystemencoding()
print updatePwd.updatePassword().decode('UTF-8').encode(type)
```

在该段代码中，为了解决中文乱码问题，在文件的开头使用了# -*- coding: UTF-8 -*-，然后导入 sys 模块，并获取文件的编码格式，从而将获取的 updatePwd 模块中的函数返回值转换成 UTF-8 编码格式。

4.6.4 运行结果

运行 call_updatePassword.py 文件，Python 解释器输出“请输入原始密码：”并等待用户输入。当用户输入的原始密码不为 python 时，结果如图 4-9 所示。如果原始密码输入正确，则提示用户“请输入新密码：”。当输入新密码的长度不符合要求时，结果如图 4-10 所示，否则提示用户“请输入确认密码：”。当两次密码不一致时，结果如图 4-11 所示。如果上述操作都正确，则结果如图 4-12 所示。

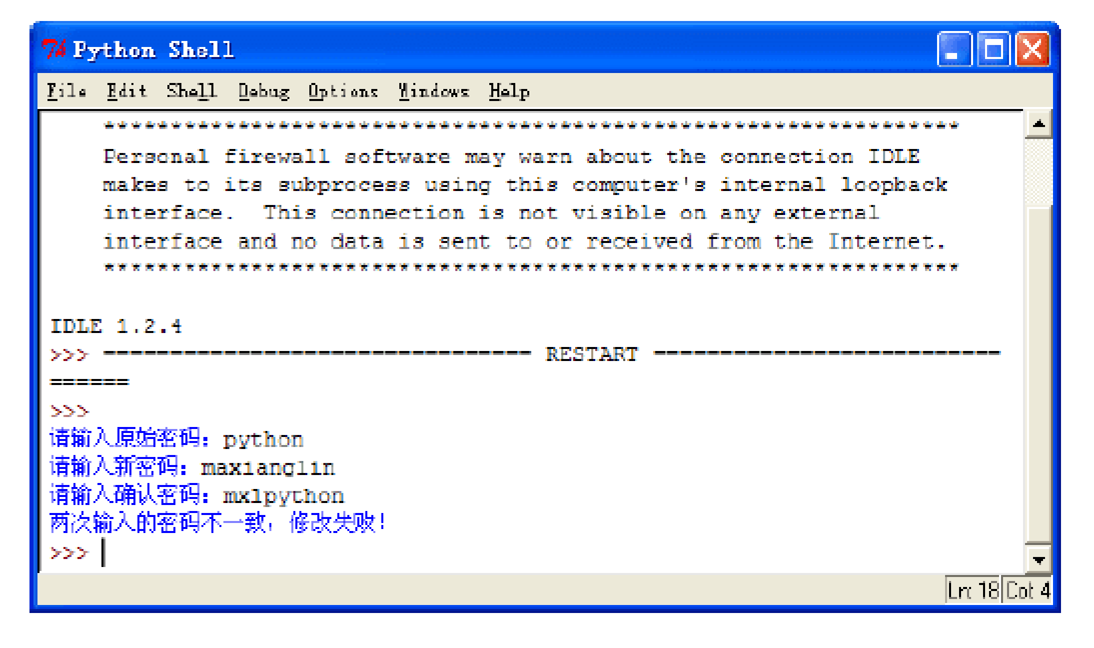


图 4-9　输入的原始密码不正确

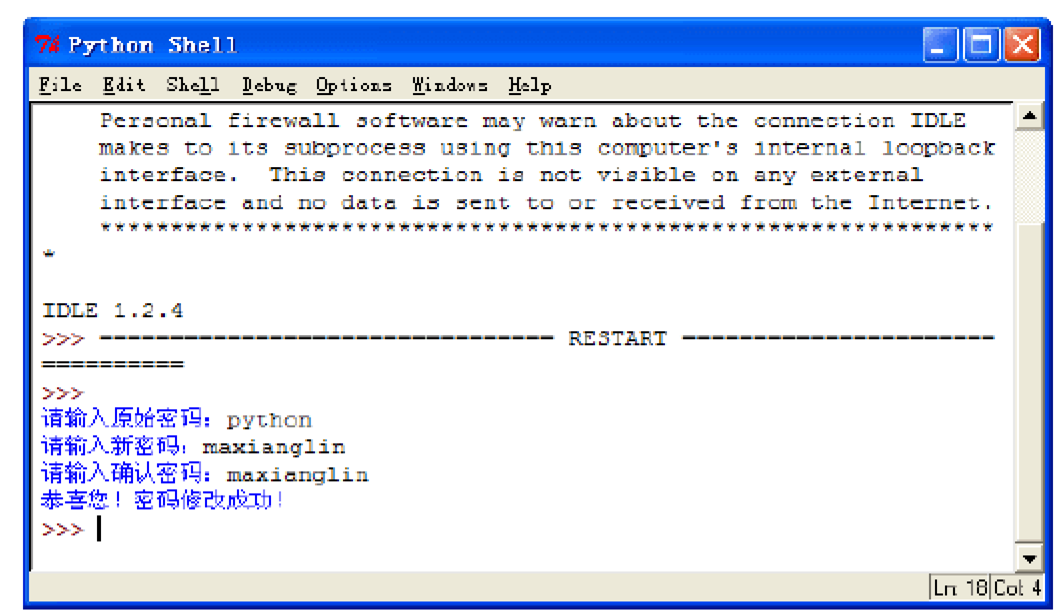


图 4-10　新密码长度不合法

图 4-11　两次密码不一致

图 4-12　密码修改成功

4.6.5　实例分析



源码解析

上述案例中，在 updatePwd 模块中定义了修改账号密码所用的函数 updatePassword()，在该函数中声明了变量 message，用于输出提示信息。把输入的新密码 newPwd 作为结果同确认密码 renewPwd 进行比较，用于判断两次密码是否一致。在文件 call_updatePassword.py 中使用 import 语句导入模块 updatePwd，然后使用 print 语句输出模块中的信息。

4.7　模 拟 购 物

模块有一些内置的属性，用于完成特定的任务。本节将针对模块的属性展开详细介绍。



视频教学：光盘/videos/04/模块属性的介绍.avi　　长度：5 分钟

4.7.1　基础知识——模块属性的介绍

Python 中的每个模块都有一些特定的属性，用于完成某项任务。下面将详细介绍 Python

中最常用的两个模块属性：__name__和__doc__。

1. __name__属性

__name__属性用于判断当前模块是不是程序的入口，如果当前程序正在被使用，__name__的值为__main__。在编写程序时，通常需要给每个模块添加条件语句，用于单独测试该模块的功能。例如，创建一个模块 myThirdModule：

```
if __name__ == '__main__':
    print 'myThirdModule 作为主程序'
else:
    print 'myThirdModule 被另一个模块调用'
```

在该段代码中，首先使用 if 条件控制语句判断__name__属性的值是否为__main__，如果成立，表明该模块为程序的入口，执行 if 块中的语句，输出“myThirdModule 作为主程序”，否则输出“myThirdModule 被另一个模块调用”。运行程序，输出结果如下：

```
myThirdModule 作为主程序
```

2. __doc__属性

Python 中的模块是一个对象，而每个对象都会有一个__doc__属性，该属性用于描述该对象的作用。下面创建另一个模块 call_myThirdModule，这个模块非常简单，只需要导入 myThirdModule 即可。

```
import myThirdModule
print __doc__
```

运行 call_myThirdModule 模块，输出结果如下：

```
myThirdModule 被另一个模块调用
```

在 call_myThirdModule.py 文件中，调用了模块的另一个属性__doc__。由于该模块没有定义文档字符串，所以输出结果为：

```
None
```



属性__doc__可以输出文档字符串的内容。

下面再来创建一个程序，具体介绍模块中的__doc__属性如何使用。

```
# 使用__doc__属性
class MyClass:
    '字符串'
    def printHello ():
        'print Hello World'
        print 'Hello World'
print MyClass.__doc__
print MyClass.printHello.__doc__
```

在该段代码中，首先定义了名称为 MyClass 的类，接着在该类中作了简单的描述，即“字符串”是对 MyClass 类的描述语句，该字符串将被保存在类的__doc__属性中。接着定义了一个名称为 printHello 的方法，在方法的主体部分，首先对该方法做了简单的描述，即 print Hello

Word 字符串就是对该方法的描述语句，该字符串将保存至 printHello()函数的__doc__属性中，在函数的最后使用了 print 语句输出 Hello World。运行该段代码，输出结果如下：

```
字符串
print Hello World
```

4.7.2　实例描述

"购物逛街"已成为现在比较流行的话语，工作了几天，大家用购物逛街的方式放松一下，这也许是一种不错的选择。假设，大商场中有这样一种好玩的机器，打开机器开始运行，首先提示"你是否购买商品？N：不买，Y：买，请输入："。当你输入 N 时，提示"您没有购买商品！"；当你输入 Y 时，提示"请输入您要购买的商品名称："，名称输入后接着提示输入该商品的数量。最后根据你所购买的商品单价，计算出购买该商品的总价格。

下面利用模块中的属性来实现上述功能。

4.7.3　实例应用

【例 4-6】模拟购物。

(1) 创建一个 Python 文件，命名为 Goods.py。

(2) 编辑购买商品所使用的 Goods 模块，在该模块中定义一个名称为 Goods 的类。首先在该类下定义一个文档字符串，用于说明该类的作用，然后在该类中定义一个实现购买商品功能的函数，并在该函数中定义一个文档字符串，用于说明是否参与购买商品活动。代码如下：

```
class Goods:
    '够买商品的活动'
    def buyGoods ():
        '是否参与该活动'
        #判断是否执行购买商品的活动
        if __name__=='__main__':
            #提示是否购买商品
            goods=raw_input('您是否购买商品？N:不买，Y:买，请输入：')
            if (goods=='Y'):
                #提示输入商品的名称及数量
                goodsName=raw_input('请输入您要购买的商品名称：')
                num=int(raw_input('请输入该商品的数量：'))
                #输出商品的名称及数量
                print '您购买的商品是：'+goodsName+ '数量：'+str(num)
                #输出并计算商品的总价
                print '您所购买的商品单价为 5 元，总价格：'+str(5*num)
            if (goods=='N'):
                print '您没有购买商品！'
        else:
            print '您放弃了参与该活动！'
```

(3) 输出已定义的文档字符串，代码如下：

```
print Goods.__doc__
print Goods.buyGoods.__doc__
```

(4) 执行是否参与购买商品活动的函数 buyGoods()。代码如下：

```
print buyGoods()
```

(5) 再次创建一个 python 文件，命名为 linkGoods.py，该文件主要是为了验证模块中的__name__和__doc__属性。代码如下：

```
import Goods
print __doc__
```

4.7.4 运行结果

运行 Good.py 文件，Python 解释器输出“您是否购买商品？N：不买，Y：买，请输入：”。如果输入 N，结果如图 4-13 所示。当输入 Y 时，结果如图 4-14 所示。根据提示信息依次输入商品名称及数量，结果如图 4-15 所示。运行 linkGoods.py 文件，结果如图 4-16 所示。

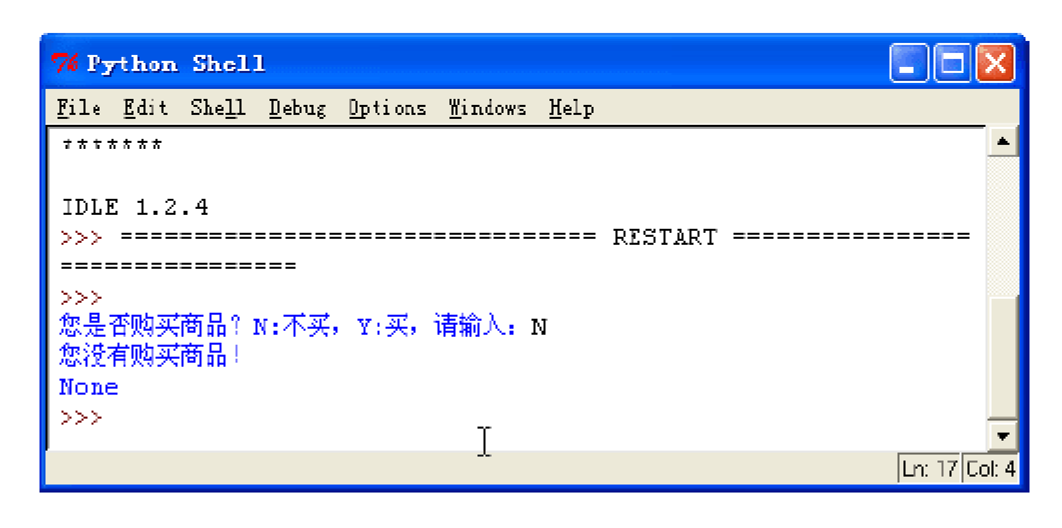


图 4-13　输入N的结果

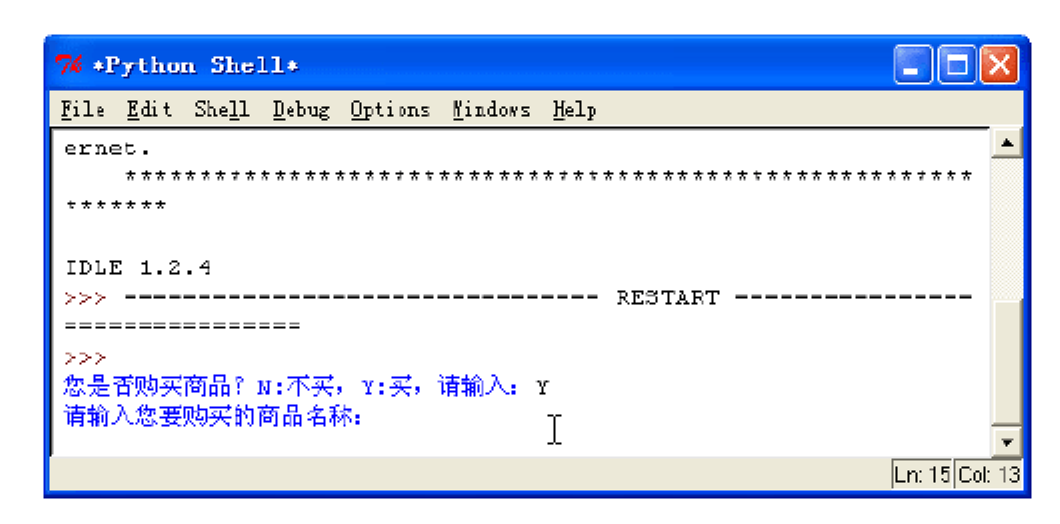


图 4-14　输入Y的结果

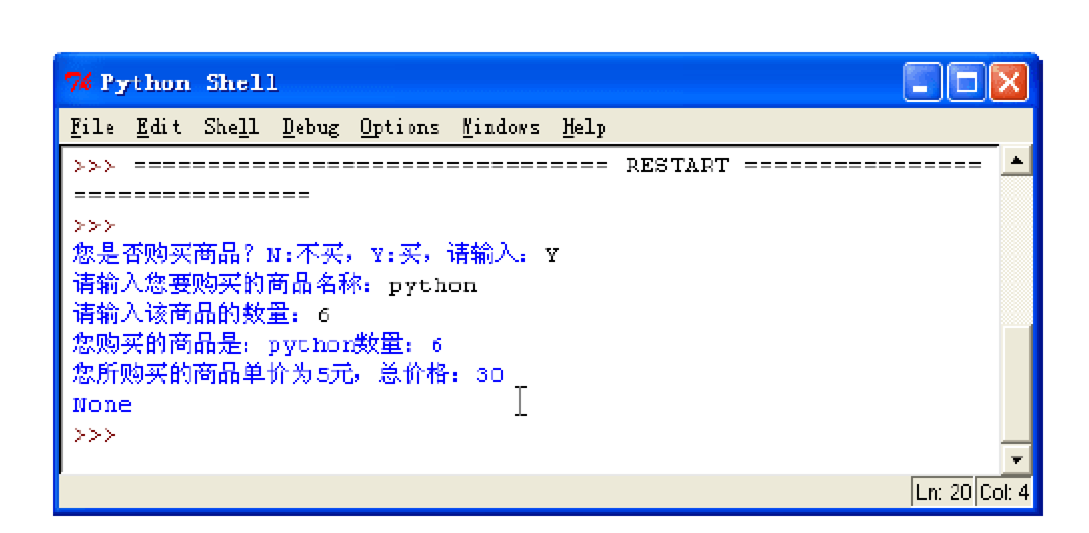


图 4-15　购买商品的结果

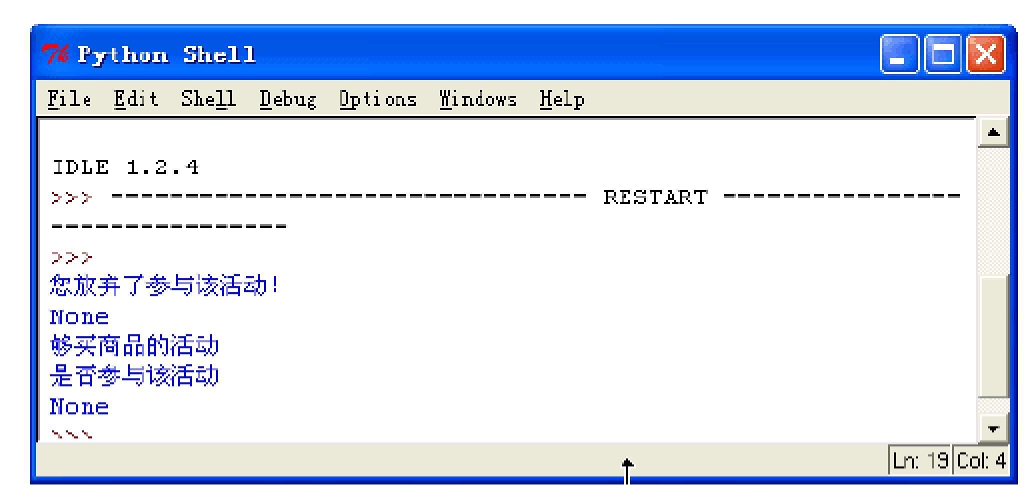


图 4-16　运行linkGoods.py的结果

4.7.5 实例分析



源码解析

上述案例中，在 Goods.py 文件中定义了一个文档字符串，该字符串的作用主要是为了测试模块中的__doc__属性。在 linkGoods.py 文件中调用 Goods 模块，用 print __doc 语句输出 Goods.py 文件中所定义的文档字符串。当满足函数 buyGoods()中的属性__name__==__main__时，执行是否购买商品的活动，否则执行 linGoods.py 文件中所输出的结果。本实例主要是为了练习模块中的__name__和__doc__两个属性。

4.8　使用模块内置函数生成验证码

Python 提供了一个内联模块——buildin，在该模块中定义了许多函数。在 Python 的程序开发中，buildin 模块中的某些函数是不可或缺的。本节将介绍几个常用的函数。

视频教学：光盘/videos/04/模块的内置函数.avi　　　　长度：13 分钟

4.8.1　基础知识——模块的内置函数

在 buildin 模块中定义了许多在软件开发过程中经常用到的函数，利用这些函数可以实现数据类型的转换、数据的计算、序列的处理等功能。下面将介绍内联模块中常用的函数。

1. apply()函数

apply()函数可以实现调用可变参数列表的功能，把函数的参数存放到一个元组或序列中。apply()函数的语法格式如下：

```
apply(function_name [ , args [ , kwargs]] )
```

参数 function_name 所表示函数的返回值就是 apply()函数的返回值，apply()函数有如下 3 个参数。

- function_name：该参数是必需的，表示自定义函数的名称。
- args：该参数是可选的，它是一个包含了自定义函数参数的列表和元组。如果不指定该参数，则表示被执行的函数没有任何参数。
- kwargs：该参数是可选的，它是一个字典类型的参数，字典中的 key 值为函数的参数名称，value 值为实际参数的值。

下面使用 apply()函数创建一个程序，具体介绍 apply()函数的使用。

```
# 定义登录函数 login()，该函数有两个参数，分别为 username 和 password
def login (username, password):
    msg = ''              # 记录返回结果的字符串
    if(username == 'admin') and (password == 'admin'):     # 验证用户名和密码
        msg = '登录成功'
    else:
        msg = '登录失败'
    return msg            #返回登录信息
# 使用 apply()函数实现调用可变参数列表
print apply(login,('admin','admin'))
```

在该段代码中，首先定义了 login()函数，同时定义了两个形式参数，然后在函数的主体部分使用 if 条件控制语句来验证用户名和密码是否合法，并将验证信息返回。最后使用 apply()函数实现调用可变参数列表的功能。因为 apply()函数中的第一个参数所表示的 login()函数的返回值为验证结果信息，即输出 apply()函数返回的结果。

```
登录成功
```



apply()函数的元组参数是有序的，例如在本案例中，元素的顺序必须与 login()函数的形式参数的顺序保持一致。

2. filter()函数

filter()函数可以对序列做过滤处理，简单地说就是用函数来过滤一个序列，把序列的每一项传递到过滤函数。对自定义函数的参数返回的结果是否为 True 做过滤，并一次性返回处理结果。如果过滤函数返回的结果为 False，那么该元素将从列表中删除该项。filter()函数的语法格式如下：

```
filter(function_name, sequence)
```

filter()函数有两个参数，其中：

- function_name：该参数是必需的，它是自定义函数，在函数 function_name()中定义过滤的规则。如果 function_name()函数的返回值为 None，则表示 sequence 序列中的每一项都为 True，从而返回左右的序列元素。
- sequence：该参数也是必需的，表示需要过滤的序列。

filter()函数的返回值由 function_name()函数的返回值决定，function_name()函数返回所有为 True 的过滤项组成的序列，返回值的类型与参数 sequence 的类型相同。例如，参数 sequence 的类型为元组，那么返回值的类型也是元组。下面使用 filter()函数创建一个程序，具体介绍 filter()函数的使用方法。

```
# 定义验证用户名的函数 validate()，长度在 4-12 之间
def validate (usernames):
    if (len(usernames) > 4) and (len(usernames) < 12):
        return usernames
# 调用 filter()，过滤 validate()函数中的元组参数
print filter(validate, ('admin','maxianglin','mxl','adm','wanglili'))
```

在该段代码中，首先定义了验证用户名的函数 validate()，然后在函数的主体部分过滤 usernames 参数中的每一项，过滤的条件为：每一项的长度在 4～12 之间。最后使用 filter()函数过滤，这里使用了('admin','maxianglin','mxl','adm','wanglili')生成待处理的元组，然后把该元组的值依次传入 validate()函数中。validate()函数返回的结果为 filter()函数的返回结果。最后 filter()函数把返回的值组成一个元组返回，输出结果如下：

```
('admin', 'maxianglin', 'wanglili')
```



filter()中的过滤函数 function_name()的参数不能为空，否则没有可存储 sequence 元素的变量，function_name()也不能处理过滤。

3. reduce()函数

reduce()函数可以实现连续处理功能。例如，对某个序列中的元素进行累加操作。reduce()函数的语法格式如下：

```
reduce(function_name, sequence[ , initial])
```

reduce()函数有 3 个参数，其中：

- function_name：该参数是必需的，表示自定义的函数名称，在函数 function_name()中实现对参数 sequence 的连续操作。
- sequence：该参数也是必需的，表示待处理的序列。
- initial：该参数是可选的，如果指定了该参数的值，则 initial 所指定的值将首先被传入 function_name()函数中进行计算。如果 sequence 参数的值为空，则对 initial 所指定的值进行处理。

reduce()函数的返回值为 function_name()函数对序列中的元素进行连续操作之后的计算结果。下面使用 reduce()函数创建一个程序，具体介绍如何使用 reduce()函数来实现对序列中的元素进行连续操作。

```
# 定义计算两个数相乘的函数 operat()
def operat(x, y):
    return x*y

# 使用 reduce()函数，对元组中的每一项进行计算，最后将计算结果返回
print reduce(operat, (1,2,3,4,5,6))
print reduce(operat, (7,8,9) , 5)
```

在该段代码中，首先定义了 operat()函数，该函数需要两个参数，执行连乘操作，并将连乘后的结果返回。下面使用 reduce()函数对元组中的元素(1,2,3,4,5,6)进行连乘，即 1*2*3*4*5*6，输出结果如下：

```
720
```

在第二次使用 reduce()函数对 operat()函数中提供的元组类型的参数执行连乘操作时，为 reduce()函数提供了第三个参数，指定值为 5，即计算 5*7*8*9 的结果，最后使用 print 语句输出计算结果。

```
2520
```



在使用 reduce()函数进行累加计算时，必须为 reduce()函数中的 function_name()函数提供两个参数，分别对应运算符两侧的操作数。

4．map()函数

map()函数可以对多个序列中的每个元素执行相同的操作，并返回一个与输入序列长度相同的列表。其中，每一个元素都是对输入序列中相应位置的元素转换的结果。map()函数的语法格式如下：

```
map(function_name, sequence[, sequence, ...])
```

其中，参数 function_name 表示自定义函数的名称，实现对序列中每个元素的操作。sequence 参数表示待处理的序列，参数 sequence 的个数可以是多个。如果传给 map()的函数参数接受多个参数，那么就可以给 map()传递多个序列，如果这些传进来的序列长度不一，那么会在短序列后面补 None。函数参数还可以是 None，这时就会用序列参数中的元素生成一个元组的序列。下面使用 map()函数创建一个程序，具体介绍 map()函数的使用方法。

```
#
def add1(a):
    return a + 1
```

```
def add2(a, b):
    return a + b
def add3(a, b, c):
    return a + b + c

a1 = [1,2,3,4,5]
a2 = [1,2,3,4,5]
a3 = [1,2,3,4,5]
```

在该段代码中，首先定义了 3 个函数，分别实现序列元素增 1 运算、两个序列元素对应相加运算和 3 个序列元素对应相加运算。接着声明 3 个列表，分别为 a1、a2 和 a3，并赋予相同的值。下面使用 map()函数来调用 add1()函数，实现序列元素增 1 运算。代码如下：

```
b = map(add1, a1)
print b
```

该段代码把 a1 表示的列表中的元素增 1，然后将计算结果组成一个列表返回，输出结果如下：

```
[2, 3, 4, 5, 6]
```

接着使用 map()函数来调用 add2()函数，实现两个序列对应元素相加运算。

```
b = map(add2, a1, a2)
print b
```

在该段代码中，将 a1 和 a2 表示的列表中的对应元素相加，然后将计算结果组成一个列表返回，输出结果如下：

```
[2, 4, 6, 8, 10]
```

最后使用 map()函数来调用 add3()函数，实现 3 个序列对应元素相加的运算：

```
b = map(add3, a1, a2, a3)
print b
```

在该段代码中，将 a1、a2 和 a3 表示的列表中的对应元素相加，然后将计算结果组成一个列表返回，输出结果如下：

```
[3, 6, 9, 12, 15]
```



如果 map()中提供了多个序列，则每个序列中的元素一一对应进行计算。如果每个序列的长度不相同，则在长度短的序列后补充 None，再进行计算。

4.8.2　实例描述

在做后台关系系统时，往往需要提供一个登录页面，当用户输入相应的用户名、密码和验证码时，才可登录到后台管理系统的主页面。验证码在登录页面中是必不可少的，它能够防止用户在很短的时间内登录多次，从而对服务器造成很大的影响。验证码通常由 4 位数字组成。

下面使用 Python 中的 filter()函数来随机生成 4 位数字，作为登录页面中的验证码。

4.8.3　实例应用

【例 4-7】使用模块的内置函数生成 4 位数字的验证码。

(1) 新建 Python 文件，命名为 filter.py。

(2) 编辑 filter.py 文件，从 random 模块中导入 randint 函数，使用该函数随机生成 1000～9999 中的任意 4 个数字。抽取其中的 10 个，然后使用 for ... in 循环将抽取的 10 个 4 位数组成一个列表。代码如下：

```
# 使用 from ... import 语句从 random 模块中导入 randint 函数
from random import randint
# 声明 allNums 变量，类型为列表类型
allNums = []
# 使用 for ... in 循环抽取从 1000-9999 随机生成的数字中的 10 个
for eachNum in range(10):
    allNums.append(randint(1000,9999))
```

(3) 使用 print 语句将获取到的 10 个 4 位数组成的列表输出，并使用 filter()函数过滤其中的偶数和奇数。代码如下：

```
print '随机从 1000-9999 生成的数字中获取的 10 个值为：'+str(allNums)
print '偶数的有：'+str(filter(lambda n:n%2==0, allNums))
print '奇数的有：'+str(filter(lambda n:n%2!=0, allNums))
```

由于 allNums 变量和 filter()函数返回的结果均为列表类型，因此需要使用 str()函数转换成字符串类型，再使用 print 语句输出。

4.8.4　运行结果

运行 filter.py 文件，输出随机生成的 1000～9999 中的 10 个数字，并输出其中的偶数和奇数，结果如下：

```
随机从 1000-9999 生成的数字中获取的 10 个值为：[4597, 3334, 6941, 3473, 9830, 1487,
8087, 7782, 5039, 5962]
偶数的有：[3334, 9830, 7782, 5962]
奇数的有：[4597, 6941, 3473, 1487, 8087, 5039]
```

4.8.5 实例分析



源码解析

在上述案例的 for ... in 循环中，使用了 append()函数将随机生成的 1000～9999 中的 10 个数字组成一个列表，然后使用 str()函数将这个列表中的所有元素输出。在判断其中有哪些数字是偶数，哪些是奇数时，使用了 filter()函数。在使用该函数时，传入的第一个参数为 lambda+表达式，其中的 lambda 是一个虚拟函数的名称，作用是动态地创建一个函数，其后只能跟表达式；传入的第二个参数为抽取的 10 个 4 位数组成的列表，从而根据 lambda 函数后紧跟着的表达式过滤该列表中的数字，从而得出哪些是偶数，哪些是奇数。

4.9 常见问题解答

4.9.1 导入Python模块引起的问题



导入 Python 模块引起的问题！

网络课堂：http://bbs.itzcn.com/thread-15826-1-1.html

我不是很了解 Python 语言，模块是怎么导入的呢？我做了一个简单的例子，导入模块竟然出错，这是我的代码：

```
def max(x,y):
    if x>y:
        print x
    else:
        print y
```

保存该段代码，并命名为 max.py，接着编辑了调用该模块的文件 call_max.py，代码如下：

```
import max
a=raw_input('请输入第一个数字：')
b=raw_input('请输入第二个数字：')
max(a,b)
```

运行代码，却出现如下错误：

```
Traceback (most recent call last):
  File "C:\Documents and Settings\Administrator\桌面\04\call_max.py", line 4,
in <module>
    max(a,b)
TypeError: 'module' object is not callable
```

这是怎么回事呢？

【解决办法】错误是你没有正确地调用模块中的函数。例如，调用模块中的 max 函数，代

码为：

```
max.max(a,b)
```

如果在 call_max.py 文件的头部使用 from ... max import *导入 max 模块中的所有函数和类，那么在文件中才可以直接使用 max(a,b)来调用模块中的函数。

4.9.2　关于Python函数不加括号的问题



关于 Python 函数不加括号的问题！
网络课堂：http://bbs.itzcn.com/thread-15827-1-1.html

下面先来看一段代码。

```
def a(x):
    def b(y):
        return x+y
    return b
```

在函数 a()中，最后的 return 语句中的 b 没有加括号，但是函数运行很正常。下面来调用一下，首先使用：

```
print a(1)(2)
```

计算出的结果为 3，运行完全正常。再使用下面的方式调用：

```
print a(6)
```

输出结果为：<function b at 0x00CD9270>。这怎么理解呢？

【解决办法】为了解释清楚这种现象，下面先来看一段代码。

```
def b(y):
    return x+y
def a(x):
    return b
```

这种写法无法将 x 传到 b()函数中，其实当调用 a 时，打印的是 a 的内存地址，a(x)调用的就是 a 的方法，返回的是 b，相当于直接打印 b 的内存地址，所以 a(x)(y)调用的 b()方法，返回 x+y 的值，这里 x 取的是 a()方法的参数值，y 是 b()方法的参数值，最后返回 b，相当于返回 b()函数对象。

4.10　习　　题

一、填空题

(1) 在 Python 语言中，横线处函数的定义需要使用__________保留字来标识。

(2) 在下面代码中，若要程序输出“添加成功！”信息，横线处应填写__________。

```
def addUser (usernames = [] , realname='无名', addres = '河南省郑州市'):
    username = usernames[__________]
```

```
    if username == 'admin':
        print "添加成功！"
    else:
        print "添加失败！"
addUser(['admin','maxianglin','yangxiaona'])
```

(3) 导入一个 MyModule 模块下的所有类和函数，可以使用的语句是：__________。

(4) __________属性用于判断当前模块是不是程序的入口。__doc__属性用于描述 Python 中的某个对象的作用。

二、选择题

(1) 下面的选项中，正确调用 addUser()函数的选项是__________。

```
def addUser (username , realname , addres = '河南省郑州市'):
    if(username != '') and (username != None):
        print '添加成功！'
    else:
        print '添加失败！'
```

A. addUser(username='admin')

B. addUser('admin')

C. addUser(username ='admin' , addres ='河南省安阳市')

D. addUser('admin' , '无名')

(2) 在程序开发过程中，常常需要传递可变长度的参数。在函数的参数前使用标识符__________可以实现这个要求。

A. ref　　B. *　　C. **　　D. 不需要输入标识符

(3) 一个模块是由代码、类和函数组成的，其中类和函数可以有__________个。

A. 1　　B. 2 个函数，一个类

C. 2 个函数，2 个类　　D. 0 个或多个

(4) 执行下面的程序代码，输出结果为：__________。

```
def filter_fun(arrays):
    if(arrays=='admin'):
        return arrays
print filter(filter_fun,('admin','admin','maxianglin','wanglili'))
```

A. None　　B. ('admin')　　C. ('admin', 'admin')　　D. 'admin', 'admin'

三、上机练习

上机练习：检测用户注册信息。

定义函数 register()，该函数的参数有 5 个，分别是 username、password、realname、sex 和 addres，在该函数的主体部分完成检测功能。检测的条件为：用户名(username)的长度必须在 6～12 之间，密码(password)的长度在 8～20 之间，真实姓名(realname)不能为空。如果检测通过，则输出“注册成功！”的信息，如图 4-17 所示。否则让用户重新输入注册信息，如图 4-18 所示。

```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.4
>>> ================================ RESTART ================================
>>>
请输入用户名：maxianglin
输入密码：maxianglin
请输入真实姓名：马向林
请输入性别：女
请输入居住地址：河南省郑州市
注册成功！
>>> |
Ln: 20 Col: 4
```

图 4-17　注册成功

```
Python Shell
File Edit Shell Debug Options Windows Help
>>> ================================ RESTART ================================
>>>
请输入用户名：mxl
输入密码：maxianglin
请输入真实姓名：马向林
请输入性别：女
请输入居住地址：河南省郑州市
输入的用户名长度只能在4-12之间，请重新输入
请输入用户名：maxianglin
输入密码：mxl
请输入真实姓名：马向林
请输入性别：女
请输入居住地址：河南省郑州市
输入的密码长度只能在8-20之间，请重新输入
请输入用户名：maxianglin
输入密码：maxianglin
请输入真实姓名：马向林
请输入性别：女
请输入居住地址：河南省郑州市
注册成功！
>>> |
Ln: 32 Col: 4
```

图 4-18　重新输入注册信息



第5章　数据结构

内容摘要

本章将引入一个新的概念：数据结构。数据结构是通过某种方式(例如对元素进行编号)组织在一起的数据元素的集合，这些数据元素可以是数字或者字符，甚至可以是其他数据结构。在 Python 中有 3 种内建的数据结构：列表、元组和字典，最基本的数据结构是序列(sequence)，序列中的每个元素被分配一个序号，即元素的位置，称为索引。

本章将详细的介绍列表、元组和字典这三大类型的数据结构的创建、使用和操作，并对序列和引用作简单介绍。

学习目标

- 掌握列表的创建、使用。
- 熟练操作列表。
- 掌握元组的创建。
- 掌握元组的访问。
- 掌握元组的操作。
- 掌握字典的创建。
- 掌握字典的访问。
- 掌握字典的方法。
- 掌握字典的操作。
- 了解序列。

5.1 Python的“苦力”——列表

在前面的章节中，已经多次使用到列表(List)，它的强大之处不言而喻。List 是处理一组有序元素的数据结构，即可以在列表中存储一个序列的元素。在 Python 中，每个元素之间用逗号分隔，列表中的元素包含在方括号中。一旦创建了列表，就可以添加、删除或搜索列表中的项目。本节将介绍 Python 中列表的创建、使用和操作方法。

视频教学：光盘/videos/05/列表的创建和使用.avi　　长度：22 分钟

视频教学：光盘/videos/05/列表的查找、排序与反转.avi　　长度：9 分钟

5.1.1 基础知识——列表的创建

List(列表)是可变的数据类型，即这种类型是可以被改变的，它由一系列的元素组成，且所有元素都包含在一对方括号中，可以在列表中添加任何类型的元素。列表的创建格式如下：

```
list_name = [element1, element2, element3,...]
```



有一定 Java 语言基础的人应该了解 Java 语言中的 List 接口，其中的 ArrayList 类继承自 List 接口，实现了动态数组功能，可以任意添加或者删除任意类型的对象。Python 中的 List(列表)与 Java 语言中的 ArrayList 类似，用法更灵活。

列表可以使用所有适用于序列的标准操作，更重要的是列表的长度是不固定的，可以随意更改。下面将介绍一些可以改变列表的方法：添加元素，元素赋值，删除元素以及分片赋值。

1. 添加元素

Python 为 List 类提供了 append()方法，这个方法用于在列表的尾部添加一个元素，该方法的声明如下：

```
append(value)
```

其中，参数 value 的类型为 object，即可以为 List 添加任何类型的元素。和 Java 语言中的 ArrayList 类一样，需要使用点号来调用 List 类中的 append()方法。下面创建一个示例，具体介绍如何使用 append()方法向一个列表添加元素。

```
# 定义一个含有 5 个元素的列表
userList = ['0001', '0002', '0003', '0004', '0005']
# 使用 len()函数获取 userList 列表中的初始个数
print '目前有学生'+str(len(userList))+'个'
print '刚来一个学生'
# 由于刚来一个学生，因此使用 append()方法向 userList 列表的尾部添加一个元素为 0006
userList.append('0006')
# 再次使用 len()函数获取当前 userList 列表中的长度
print '现有学生'+str(len(userList))+'个，他们是：'
for item in userList:
```

```
    print item
```

在该段代码中，使用到一个 len()函数，该函数用于获取一个对象的长度，这里用于获取列表 userList 的长度。在代码的最后使用了 for … in 循环遍历 userList 列表，将列表中的元素遍历输出。运行该段代码，输出结果如下：

```
目前有学生 5 个
刚来一个学生
现有学生 6 个，他们是：
0001
0002
0003
0004
0005
0006
```

其实，Python 中还有一个方法用于将元素插入到列表中的指定索引位置，该方法为 insert()。insert()方法的语法格式如下：

```
insert(index, value)
```

该方法有两个参数：第一个参数 index 为将要插入到列表中的元素指定索引位置，第二个参数 value 为要插入的值。下面使用 insert()方法创建一个示例，具体了解该方法与 append()方法之间的区别。

```
userList = ['0001', '0002', '0003', '0004', '0005']
print '目前有学生'+str(len(userList))+'个'
# 使用 insert()方法在列表的索引为 2 的位置上插入 0006 的值
userList.insert(2,'0006')
print '现有学生'+str(len(userList))+'个，他们是：'
for item in userList:
    print item
```

在该段代码中，使用了 List 类中的 insert()方法在 userList 列表中的索引为 2 的位置上插入一个值为 0006 的元素，最后使用 for … in 循环遍历 userList 集合，输出元素。运行该段代码，输出结果如下：

```
目前有学生 5 个
现有学生 6 个，他们是：
0001
0002
0006
0003
0004
0005
```

2. 元素赋值

改变列表是一件很容易的事情，只需要使用在前面章节中提到的普通赋值语句即可。然而，并不是给一个变量赋值，而是需要给列表中的元素赋值，因此需要使用索引标记来为某个特定的位置明确的元素赋值，赋值的语法格式如下：

```
list_name[index] = value
```

其中，方括号中的 index 表示列表的索引，列表的索引从 0 开始，比如第 1 个元素的索引

为 0，第 2 个元素的索引为 1，第 3 个元素的索引为 2，依此类推。value 为元素的值，该值可以是任意类型的(包括字典类型、元组类型及其他特殊类型)。



不能为一个位置不存在的元素进行赋值，如果列表的长度为 2，那么不能为索引为 3 的元素进行赋值。如果需要这样做，那就必须创建一个长度为 4(或者更长)的列表。

```
# 定义列表 userList
userList = ['0001', '0002', '0006', '0004', '0005']
print '初始化的 userList 列表为：'+str(userList)
# 修改列表的第 3 个元素的值，由 0006 改为 0003，索引从 0 开始，第 3 个元素的索引为 2
userList[2] = '0003'
# 输出修改后的列表
print '更新后的 userList 列表为：'+str(userList)
```

在该段代码中，首先定义了一个含有 5 个元素的列表，其中第 3 个元素的值为 0006，接着使用索引方式将第 3 个元素的值改为 0003，最后将修改后的列表输出。运行该段代码，输出结果如下：

```
初始化的 userList 列表为：['0001', '0002', '0006', '0004', '0005']
更新后的 userList 列表为：['0001', '0002', '0003', '0004', '0005']
```

3. 删除元素

列表的删除可以使用 remove()方法，该方法用于移除列表中指定值的第一个匹配值，如果要删除的元素值不存在，Python 程序将抛出 ValueError 异常。remove()方法的语法格式如下：

```
remove(value)
```

其中，value 参数表示要删除的列表中指定的元素值。下面创建一个示例，具体了解一下该方法的使用。

```
# 定义一个含有 5 个元素的列表
userList = ['0001', '0002', '0003', '0004', '0005']
print '初始化的 userList 列表为：'+str(userList)
# 使用 remove()方法将元素值为 0003 的元素删除
userList.remove('0003')
# 输出删除之后的列表
print '删除一个元素后的 userList 列表为：'+str(userList)
```

在该段代码中，首先定义了一个长度为 5 的列表，然后使用 List 类中的 remove()方法将第 3 个值删除，最后输出删除之后的结果。运行该段代码，输出结果如下：

```
初始化的 userList 列表为：['0001', '0002', '0003', '0004', '0005']
删除一个元素后的 userList 列表为：['0001', '0002', '0004', '0005']
```



如果 List 列表中存在两个相同的元素，此时调用 remove()方法移除同名元素，将只能删除 List 列表中位置靠前的元素。例如，如果 List 中存在两个 0003 元素，执行语句 userList.remove('0003')之后，其中一个 0003 将被删除，而且此元素是首次出现的那个 0003 元素。

从一个列表中删除元素还可以使用 del 语句来实现，该语句将删除列表中指定索引位置所

表示的元素。del 语句的格式如下：

```
del list_name[index]
```

其中，index 表示将要删除的元素所对应的索引位置。下面使用 del 语句来删除 userList 列表中的第 2 个元素，即索引为 1 的元素。修改上面的代码为：

```
userList = ['0001', '0002', '0003', '0004', '0005']
print '初始化的 userList 列表为：'+str(userList)
# 使用 del 语句删除列表中的第 2 个元素
del userList[1]
print '删除第 2 个元素后的 userList 列表为：'+str(userList)
```

在该段代码中，使用了 del 语句将索引为 1 的元素删除，即列表中的第 2 个元素，然后将删除之后的列表输出。运行该段代码，输出结果如下：

```
初始化的 userList 列表为：['0001', '0002', '0003', '0004', '0005']
删除第 2 个元素后的 userList 列表为：['0001', '0003', '0004', '0005']
```

4. 分片赋值

分片(slice)是列表的一个子集，分片是从第 1 个索引到第 2 个索引(不包含第 2 个索引所指向的元素)所指定的所有元素。分片索引可以为正数或负数，两个索引之间用冒号分隔。分片的格式如下：

```
list_name[m : n]
```

其中，m、n 可以是正整数或负整数。分片索引与元素的对应关系如图 5-1 所示。其中，list_name[2 : 5]将返回(value3, value4, value5, value6)。

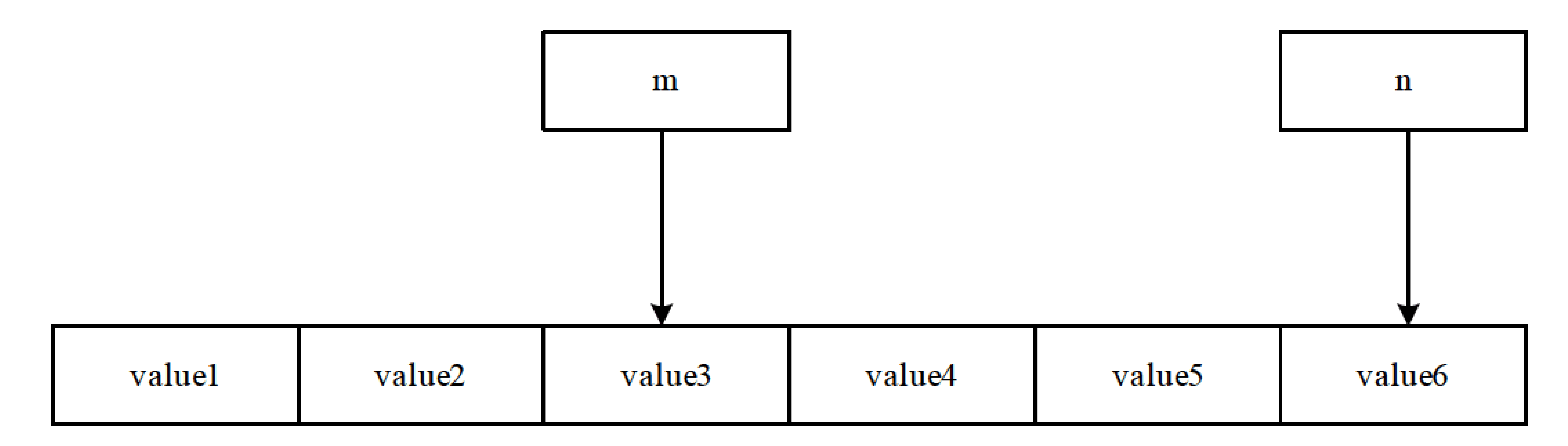


图 5-1　分片索引计数

分片是一个非常强大的特性，分片赋值则更加强大。下面来看一个例子。

```
# 使用 list()方法，定义一个含有 6 个元素的列表
userList = list('Python')
print userList
# 使用冒号分片，从第 3 个元素开始到结束，赋值为 rite
userList[2:]=list('rite')
print userList
```

在该段代码中，首先使用 list()方法定义一个含有 6 个元素的列表，元素分别为 P、y、t、h、o 和 n，然后使用了冒号将该列表分片，从列表的第 3 个元素到结束，将原有的元素变为 r、i、

t 和 e，最后将赋值后的列表输出。运行该段代码，输出结果如下：

```
['P', 'y', 't', 'h', 'o', 'n']
['P', 'y', 'r', 'i', 't', 'e']
```

还可以一次为多个元素赋值，可以使用与原序列不等长的序列将分片替换。

```
# 定义一个含有 6 个元素的列表
userList = list('Python')
# 从列表的第 2 个元素开始到结束
userList[1:] = list('rite')
print userList
```

分片赋值语句可以在不需要替换任何原有元素的情况下插入新的元素。

```
# 定义一个含有两个元素的列表
numbers = [0,6]
# 使用分片，替换原有的空的分片
numbers[1:1]=[1,2,3,4,5]
print numbers
```

该段代码只是“替换”了一个空的分片，实际操作是插入一个序列，依此类推。通过分片赋值来删除元素也是可行的。

5.1.2 基础知识——列表的使用

列表的使用非常简单，支持索引、分片以及多元列表等特性，但是列表中的元素可以修改，而且存在一些处理列表的方法。

1. 使用负索引访问列表元素

列表的索引有一些特殊的用法，这些用法对获取列表元素值非常便捷。其中，负索引就是列表的特殊索引。负索引从列表的尾部开始计数，最尾端的元素索引表示为-1，次尾端的元素索引为-2，依此类推。负索引与元素的对应关系如图 5-2 所示。

下面使用负索引来访问列表元素，具体介绍负索引的使用方法。

```
userList = ['0001', '0002', '0003', '0004', '0005', '0006']
print userList[-2]
```

在该段代码中，使用了索引为-2 的值来访问列表从尾部开始计数的第 2 个数，输出结果如下：

```
0005
```

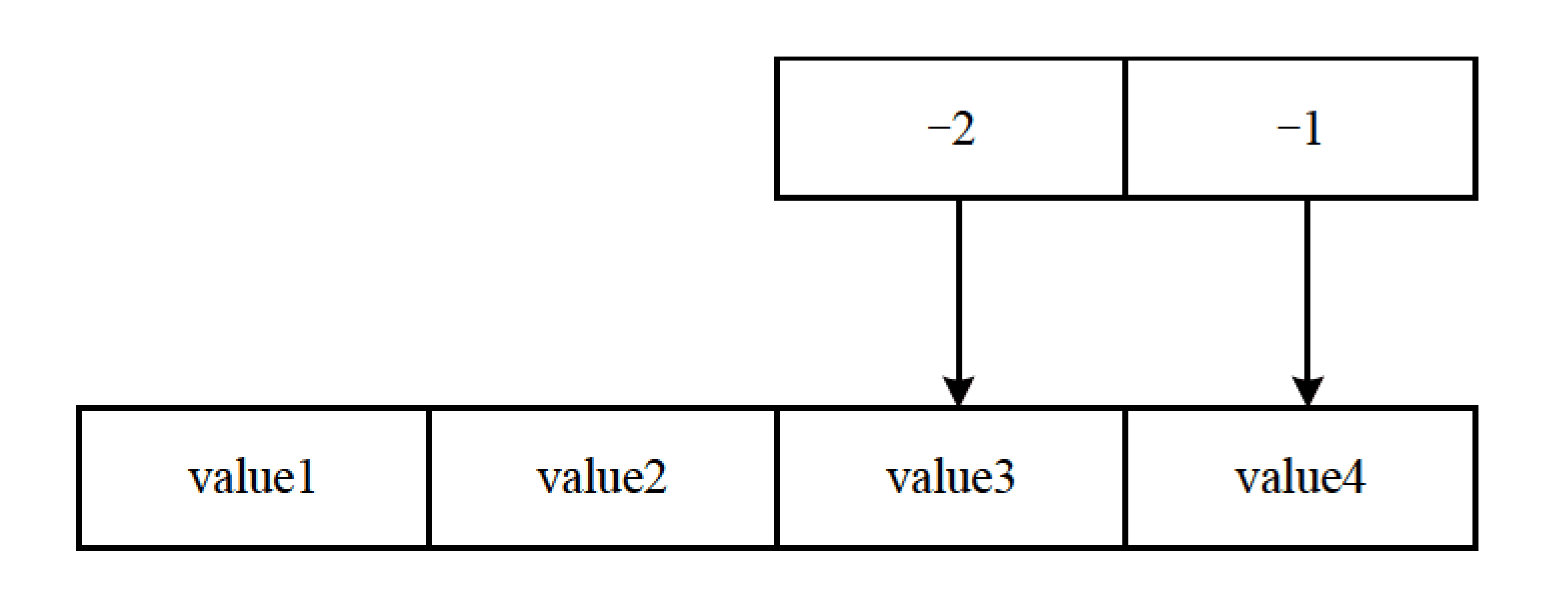


图 5-2　负索引计数

2. 列表的分片处理

与使用索引来访问单个元素类似，可以使用分片操作来访问一定范围内的元素。分片通过相隔的两个索引来实现。

```
# 定义一个含有 6 个元素的列表
userList = ['0001', '0002', '0003', '0004', '0005', '0006']
# 获取索引为 2、3 和 4 的元素值
subUser1= userList[2:5]
# 获取索引为-3 和-2 的元素值
subUser2= userList[-3:-1]
# 获取索引为 0、1 和-1 的元素值
subUser3= userList[0:-2]
print subUser1
print subUser2
print subUser3
```

在该段代码中，首先声明了一个含有 6 个元素的列表，然后使用分片处理 userList[2:5]获取列表中索引为 2、3 和 4 的元素值(不包括索引为 5 的元素)，接着使用 userList[−3: −1]获取索引为−3 和−2 的元素值，紧接着使用 userList[0:−2]获取索引为 0、1 和−1 的元素值，最后使用 print 语句将分片处理后的 subUser1、subUser2 和 subUser3 变量输出。运行该段代码，输出结果如下：

```
['0003', '0004', '0005']
['0004', '0005']
['0001', '0002', '0003', '0004']
```

3. 遍历二元列表

列表还可以由其他列表组成。例如，二元列表可以表示为：

```
list_name = [[element1, element2,...],[element3, lement4, ...],...]
```

list_name 列表是一个二元列表，该列表由多个子列表组成。下面定义一个二元列表。

```
# 定义列表 1
userList1=['0001', '0002', '0003', '0004']
# 定义列表 2
userList2=['0005', '0006', '0007']
#定义二元列表，元素由列表 1 和列表 2 组成
userList=[userList1, userList2]
print userList
```

上述代码首先定义了一个含有 4 个元素的列表，名称为 userList1，接着定义了一个含有 3 个元素的列表，名称为 userList2，然后定义了一个名称为 userList 的二元列表，元素由 userList1 和 userList2 列表组成。最后使用 print 语句将二元列表输出。输出结果如下：

```
[['0001', '0002', '0003', '0004'], ['0005', '0006', '0007']]
```

定义一个二元列表很容易，那么如何访问二元列表呢？访问二元列表的语法格式如下：

```
list_name[index1][index2]
```

其中，index1 为二元列表中的元素索引，index2 为要访问的二元列表中 index1 所指向的列表中的元素索引。下面创建一个示例，具体介绍如何访问二元列表。在上面代码的基础上，添加下列代码：

```
print "userList[0][1]=",userList[0][1]
print "userList[1][0]=",userList[1][0]
print "userList[0][3]=",userList[0][3]
```

在第 1 行代码中，userList[0][1]表示需要访问 userList 列表中的第 1 个元素['0001', '0002', '0003', '0004']中的第 2 个元素，输出结果为：

```
userList[0][1]= 0002
```

在第 2 行的代码中，userList[1][0]表示需要访问 userList 列表中的第 2 个元素['0005', '0006', '0007']中的第 1 个元素，输出结果为：

```
userList[1][0]= 0005
```

在第 3 行的代码中，userList[0][3]表示需要访问 userList 列表中的第 1 个元素['0001', '0002', '0003', '0004']中的第 4 个元素，输出结果为：

```
userList[0][3]= 0004
```



如果访问的二元列表中的元素不存在，比如访问 userList[1][3]，而 userList 列表中的第 2 个元素['0005', '0006', '0007']中不存在索引为 3 的值，即会抛出错误信息 IndexError: userList index out of range。

userList 列表的存储结构如图 5-3 所示。

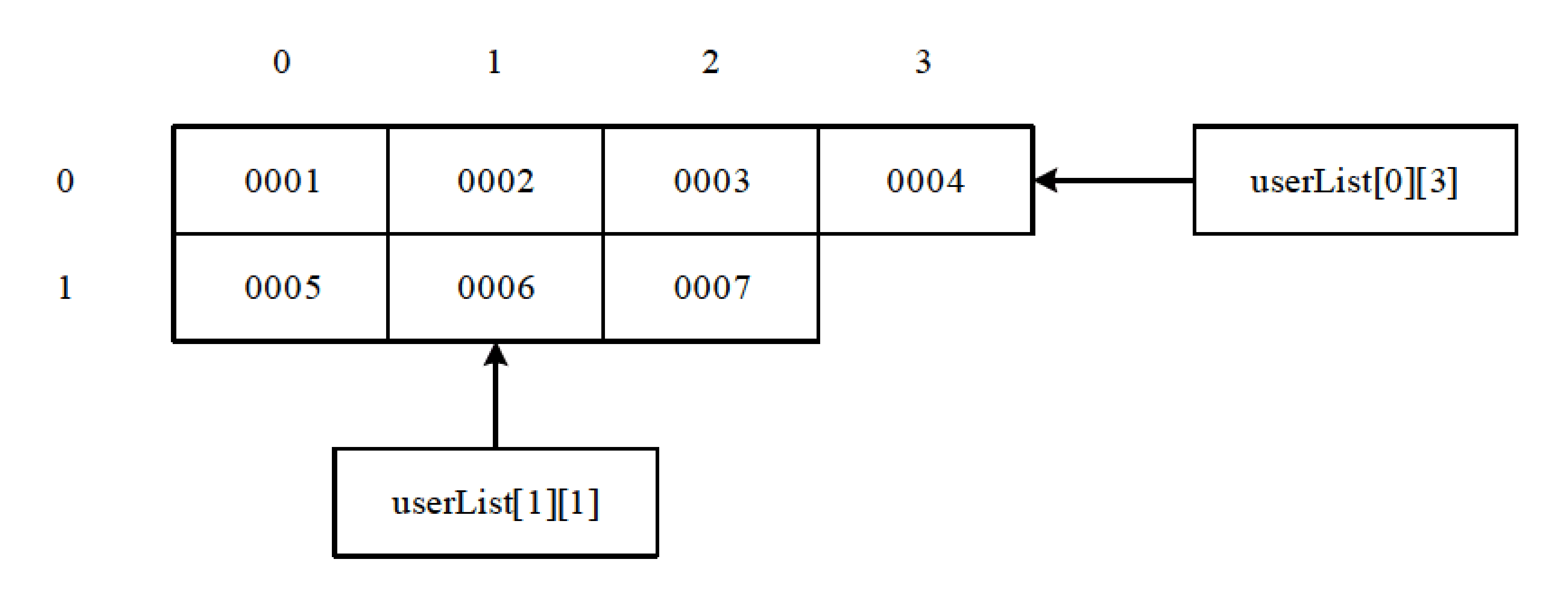


图 5-3　userList列表的存储结构图

在程序开发中，经常需要将列表中的元素全部遍历输出，尤其是二元列表，而不是单单访问列表中的某个数据。下面编辑代码，使用 for ... in 循环遍历 userList 二元列表。

```
# 循环遍历 userList 二元列表，循环两次
for i in range(len(userList)):
    #嵌套循环，内循环遍历 userList 二元列表中的每个元素
    for j in range(len(userList[i])):
        print 'userList['+str(i)+']['+str(j)+']=',userList[i][j]
```

该段代码实现了遍历 userList 二元列表的功能，最后将 userList 列表中的各个元素输出。在 userList 列表中，各子列表的长度是不同的，但这并不妨碍 userList 列表的遍历，最后输出结果为：

```
userList[0][0]= 0001
userList[0][1]= 0002
userList[0][2]= 0003
userList[0][3]= 0004
userList[1][0]= 0005
userList[1][1]= 0006
userList[1][2]= 0007
```

4．列表的连接

列表实现了连接操作的功能，实现列表的连接有两种方式：一种是调用 extend()方法连接两个不同的列表，另一种是使用运算符+、+=或*。

首先演示使用 extend()方法连接两个不同的列表。extend()方法可以在列表的末尾一次性追加另一个列表中的多个元素，即可以用新列表扩展原有的列表。代码如下：

```
userList1=['0001', '0002', '0003', '0004']
userList2=['0005', '0006', '0007']
userList1.extend(userList2)
print userList1
```

使用 extend()方法连接两个列表非常简单，就是在 userList1 列表的基础之上再添加 userList2 列表中的元素，最后输出结果为：

```
['0001', '0002', '0003', '0004', '0005', '0006', '0007']
```

可以看出：extend()方法修改了被扩展的列表(userList1)，并非真正连接了两个列表，它会返回一个全新的列表。接着演示使用+号来连接两个不同的列表，代码如下：

```
userList3=['0008', '0009', '0010']
userList4=['0011', '0012', '0013']
userList5=userList3+userList4
print userList5
```

使用+号来连接列表与连接字符串相同，就是将两个列表合并为一个新的列表，输出结果如下：

```
['0008', '0009', '0010', '0011', '0012', '0013']
```

下面再演示使用+=号来连接列表，代码如下：

```
userList5 += ['0014']
print userList5
```

使用+=号连接列表与 List 类中的 append()方法的功能相同，就是在原有的列表基础之上再添加多个元素，输出结果如下：

```
['0008', '0009', '0010', '0011', '0012', '0013', '0014']
```

最后演示使用*号来连接列表，代码如下：

```
userList6= ['0015', '0016']*2
print userList6
```

*号即乘号，['0015', '0016']*2 也就是将列表['0015', '0016']中的元素添加一倍，输出结果如下：

```
['0015', '0016', '0015', '0016']
```

5.1.3 基础知识——列表的查找、排序与反转

List(列表)除了可以进行添加、删除、修改等操作之外，Python 还提供了查找元素，为列表中的元素排序以及反转列表中的元素的功能。下面具体介绍这几个功能的实现过程。

1. 列表的查找

在 Python 中，可以通过两种方式来查找 List 中的元素：一种是使用 index()方法返回元素在列表中的位置，index()方法用于从列表中找到某个值的第一个匹配项的索引位置；另一种方法是使用保留字 in 来判断元素是否在列表中。下面创建一个示例，演示列表的查找方法。

```
userList = ['0001', '0002', '0003', '0004', '0005', '0006']
print '元素 0002 对应的索引值为：',userList.index('0002')
print '0002' in userList
```

在该段代码的第 2 行，使用 userList.index('0002')返回元素 0002 所对应的索引值，输出结果如下：

```
元素 0002 对应的索引值为： 1
```

在代码的最后使用了 in 保留字判断元素 0002 是否在 userList 列表中，输出结果如下：

```
True
```

2. 列表的排序

列表的排序需要使用 List 类中的 sort()方法。sort()方法用于在原始位置对列表进行排序。在原始位置排序意味着要改变原有的列表，从而让其中的元素按一定的顺序排列，而不是简单地返回一个已排序的列表副本。下面来看一段代码。

```
userList = ['0001', '0004', '0006', '0002', '0005', '0003']
# 调用 sort()方法，为 userList 中的所有元素排序
userList.sort()
print userList
```

在该段代码中，首先定义一个含有 6 个元素的列表，然后使用 sort()方法将列表中的元素重新排序(默认为升序)，sort()方法返回的是已经排好序的 userList 列表。运行该段代码，输出结果如下：

```
['0001', '0002', '0003', '0004', '0005', '0006']
```



使用 userList1=userList.sort()语句将列表排序后赋值给 userList1，输出结果为 None，因为 sort()方法只是修改了原有的 userList 列表，但返回的是空值，最后得到的是已排序的 userList 列表以及值为 None 的 userList1。

sort()方法有一个可选的参数 reverse，该参数是简单的布尔值(True 或 False)，用来指明列表是否要进行反向排序。修改上面的代码为：

```
userList = ['0001', '0004', '0006', '0002', '0005', '0003']
# 指定 sort()函数的 reverse 参数，设置为 True，表示需要反向排序，即降序排列
userList.sort(reverse=True)
print userList
```

在该段代码中，调用了列表的 sort()方法，并指定 reverse 的值为 True，表明需要反向排序，即需要降序排列(默认为升序)。输出结果如下：

```
['0006', '0005', '0004', '0003', '0002', '0001']
```

3．列表的反转

reverse()方法将列表中的元素反向存放，下面来看一段代码。

```
userList = ['0001', '0004', '0006', '0002', '0005', '0003']
# 调用 List 类中的 reverse()方法，将列表中的元素反转
userList.reverse()
print userList
```

在该段代码中，使用了 reverse()方法将列表中的元素反转过来。运行该段代码，输出结果如下：

```
['0003', '0005', '0002', '0006', '0004', '0001']
```



reverse()方法改变了列表但不返回值，与 sort()方法相同。

5.1.4　基础知识——用列表实现堆栈

堆栈和队列是数据结构中较为常用的结构，使用列表的 append()和 pop()方法可以模拟这两种数据结构。在前面已经讲解过 append()方法，这里不再累赘。在此，仅详细介绍 pop()方法。

pop()方法会移除列表中的元素(默认为最后一个)，并且返回该元素的值。

```
userList = ['0001', '0004', '0006', '0002', '0005', '0003']
# 使用 pop()方法，移除列表中最后一个元素(默认)
userList.pop()
print userList
```

```
# 继续移除列表中的第 1 个元素，即索引为 0 的元素
print userList.pop(0)
print userList
```

在该段代码中，第一次使用 pop()方法时，移除了 userList 列表中的最后一个元素，输出结果为：

```
['0001', '0004', '0006', '0002', '0005']
```

第二次使用 pop()方法，并指定要移除的元素所对应的索引为 0，即继续移除列表中的第一个元素，并输出要移除的元素值以及在已经移除最后一个元素的 userList 列表的基础上再次移除第一个元素之后的 userList 列表。

```
0001
['0004', '0006', '0002', '0005']
```



pop()方法是唯一一个既能修改列表又能返回元素值(除了 None 值)的列表操作方法。

使用 pop()方法可以实现一种常见的数据结构——堆栈，栈的原理就像堆放盘子一样，只能在顶部放一个盘子。同样，也只能从顶部一个一个地将盘子拿走。堆栈使得最先进入堆栈的元素最后才输出，符合“后进先出”的原则。

上述两个栈的操作(放入和移出)称为入栈和出栈。Python 没有提供入栈的方法，但可以使用 append()方法来代替。调用 append()方法可以将一个元素添加到堆栈的顶部，调用 pop()方法则可以把堆栈中的最后一个元素“弹”出来。假设现在有一个堆栈['0001', '0002', '0003', '0004']，要向堆栈中添加一个新的元素 0005，那么列表实现堆栈的原理如图 5-4 所示。

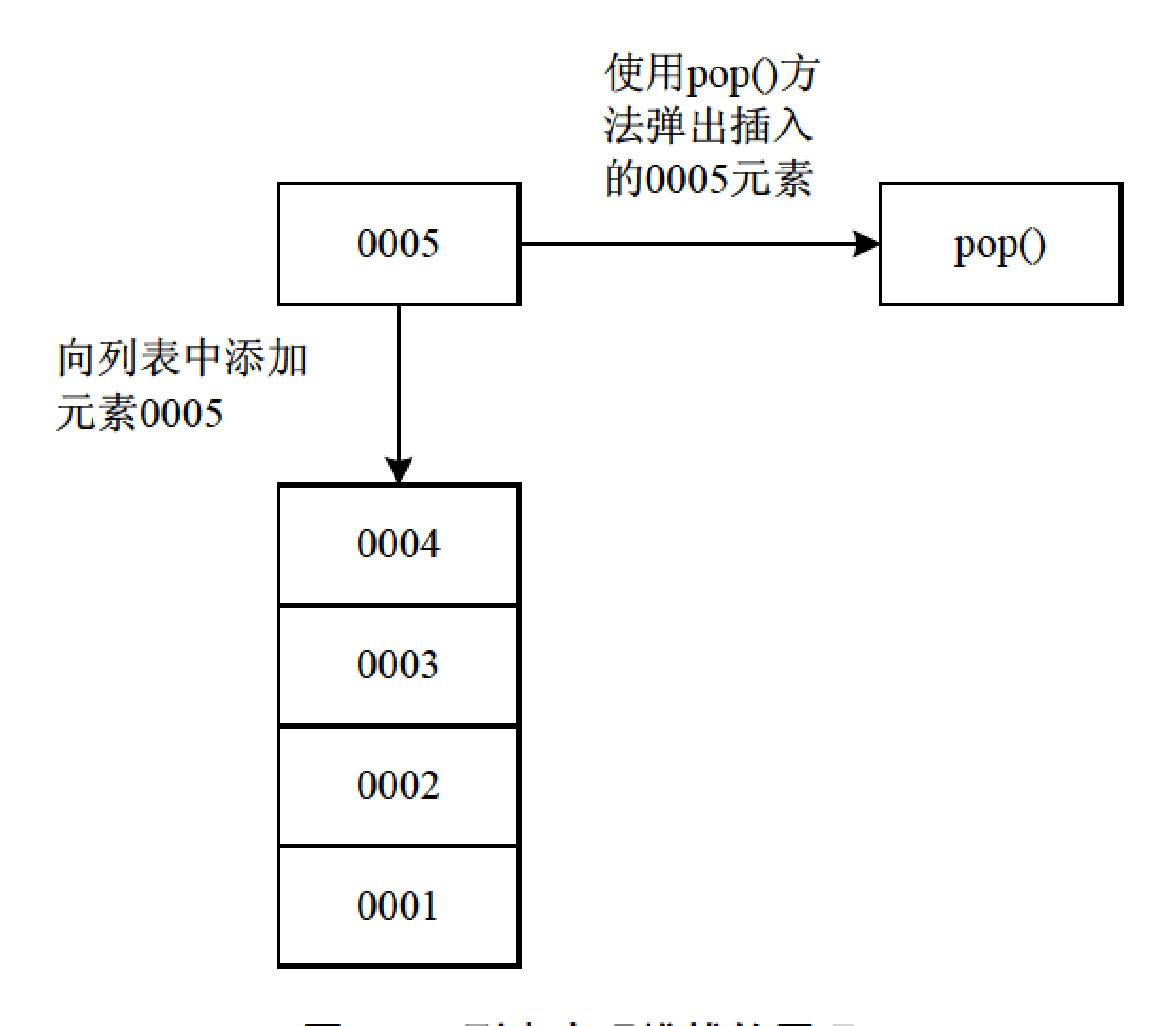


图 5-4　列表实现堆栈的原理

从图 5-4 中可以看出：0001 元素是列表中的第 1 个进入的元素，所以在堆栈的最低端，依次进入列表的元素为 0002、0003 和 0004。调用 append()方法将元素 0005 添加到列表中。append()方法将要添加的元素插入到了堆栈的顶端，此时堆栈中有 5 个元素，分别为 0001、0002、0003、0004 和 0005，然后使用 pop()方法弹出顶端的元素 0005，再依次弹出 0004、0003、0002 和 0001，达到“后进先出”的效果。下面演示一下用列表模拟堆栈的方法。

```
# 定义一个列表
userList = ['0001', '0002', '0003', '0004']
# 向列表中添加一个元素为 0005
userList.append('0005')
print '列表顺序为：'
for item in range(len(userList)):
    print userList[item]
print '取出顺序为：'
for item in range(len(userList)) :
    # 调用 pop()方法依次从上到下弹出元素
    print userList.pop()
```

在该段代码中，首先创建一个 userList 堆栈，然后使用 append()方法向堆栈中放入一个值为 0005 的元素，接着循环遍历集合，输出列表中的元素。最后使用一个 for ...in 循环遍历输出 userList 堆栈中的元素。在循环体内，使用了 pop()方法将元素从上到下弹出堆栈。运行该段代码，输出结果如下：

```
列表顺序为：
0001
0002
0003
0004
0005
取出顺序为：
0005
0004
0003
0002
0001
```

5.1.5　实例描述

我一个朋友是开水果店的，因为水果的种类太多，并且每天总是需要不停地添加新品种，所以经常需要查询水果所在放置柜中的第几个位置，以便迅速为顾客提供方便。今天我闲来无事，来他的店转悠，就想用 Python 语言为他做一个系统，这样他就不用每天在本子上记来记去了。下面就来一起做一下吧！

5.1.6 实例应用

【例 5-1】过滤列表中的重复数据。

(1) 新建 Python 文件，并命名为 repeat.py。

(2) 在 repeat.py 文件中编辑函数，用于过滤列表中的重复数据。在函数的主体部分，使用 for ... in 循环过滤列表，并将无重复的放入 checked 列表，然后使用 sort()方法将过滤后的列表 checked 降序排列，接着使用 for ... in 循环将过滤后的列表中的数据输出，最后将过滤后的列表返回。代码如下：

```
def fruitFun (fruitList):
    checked = ['香蕉','橘子','香梨']
    for e in fruitList:
        if e not in checked:
            checked.append(e)
    print '-------------欢迎使用水果信息管理系统--------------'
    print '现有水果：'
    checked.sort(reverse=True)
    for item in checked :
        print item
    return checked
```

(3) 调用 fruitFun()函数。首先声明一个列表为 fruits，用户可以输入水果名称向列表中添加数据。接着调用 fruitFun()函数，将添加水果后的列表作为参数传递至该函数中，并获取过滤后的列表。代码如下：

```
fruits = ['苹果','香蕉','橘子','香梨','橙子','苹果','香蕉','橘子']
addFruit = raw_input('请输入你要添加的水果：')
fruits.append(addFruit)
fruitList = fruitFun(fruits)
```

(4) 输入要查找的水果名称，查询该水果所在放置柜中的第几个位置。代码如下：

```
selectFruit= raw_input('请输入要查找的水果名称：')
print '你要查找的水果名称为【'+selectFruit+'】的水果在列表中的位置为第
'+str(fruitList.index(selectFruit)+1)+'个'
```

(5) 保存 repeat.py 文件。

5.1.7 运行结果

运行 repeat.py 文件，首先提示用户输入要添加的新品种水果的名称。当用户输入任意一种水果名称之后，按 Enter 键，将输出过滤重复数据之后的列表数据。然后再次提示用户输入要查找的水果名称。当用户输入任意一种水果名称之后，系统将查找出该水果所在放置柜中的位置。运行结果如图 5-5 所示。

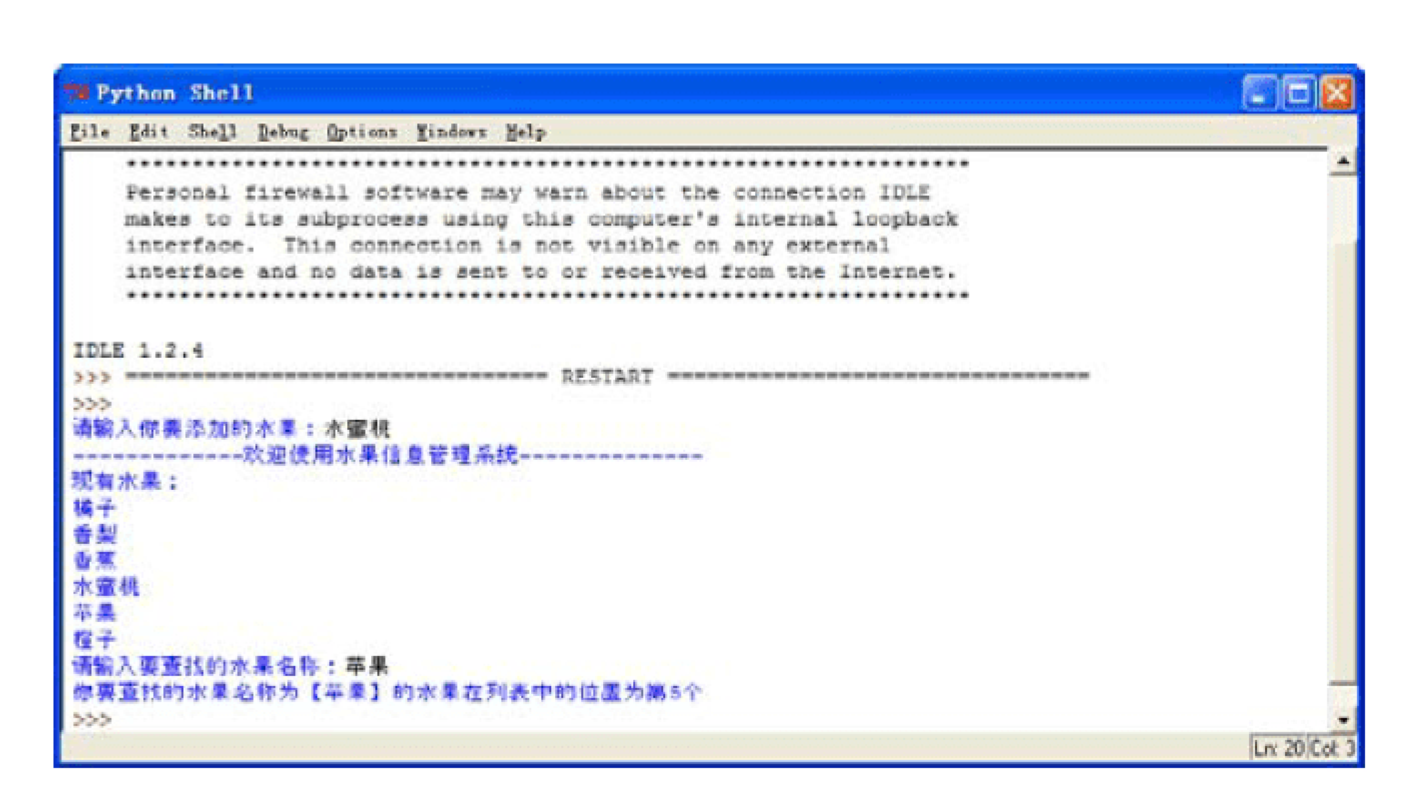


图 5-5　过滤列表中重复数据的运行结果

5.1.8　实例分析


源码解析

在上述案例中，在 fruitFun()函数中首先声明了一个名称为 checked 的列表，然后使用 for ... in 循环遍历传递过来的列表参数。在循环体内使用了 if ... not in 条件判断语句检测传递过来的列表是否在 checked 列表中不存在，如果不存在则添加至 checked 列表中，否则不添加，这样就实现了过滤的功能。

5.2　不可变序列——元组

元组和列表十分相似，元组和字符串一样是不可更变的。元组由不同的元素组成，每个元素可以存储不同类型的数据，例如字符串、数字和元组。元组通常代表一行数据，而元组中的元素则代表不同的数据项。

 **视频教学：**光盘/videos/05/元组.avi　　　　长度：18 分钟

5.2.1　基础知识——元组的创建

元组和列表一样是由一系列元素组成的，不同的是元组中的元素被包含在一对圆括号中，元素之间也用逗号隔开。元组通常使用在使语句或用户定义的函数能够安全地采用一组值的情况下，并且被使用的元组的值不能改变。

1．创建元组

创建元组时可以不指定元素的个数，相当于不定长的数组，但是一旦创建就不能修改元组的长度。创建元组的语法很简单，其格式如下：

```
tuple_name = (element1, element2, element3, ...)
```

空元组可以使用没有包含内容的两个圆括号来表示。

```
tuple_name = ()
```

如果创建的元组只包含一个元素，那么该元素后面的逗号是不可忽略的。如果忽略了逗号，Python 将无法识别变量 tuple_name 是元组还是表达式，会误认为圆括号中的内容为表达式。下面来看一段代码。

```
>>> (42)
42
>>> (42,)
(42,)
```

如果把一个元素后面的逗号忽略，那么 Python 会认为这是一个表达式，而非元组，因此如果创建的元组中只有一个元素，那么该元素的语法格式如下：

```
tuple_name = (element, )
```

下面创建一个名称为 userTuple 的元组，该元组由 6 个元素组成，元素之间使用逗号隔开，代码如下：

```
userTuple = ('0001', '0002', '0003', '0004', '0005', '0006')
print userTuple
```

在该段代码中，只是简单地初始化 userTuple 元组，在 userTuple 元组中含有 6 个元素，元素的类型为字符串。该元组的长度为 6，是不可改变的。运行该段代码，输出结果如下：

```
('0001', '0002', '0003', '0004', '0005', '0006')
```



元组的索引与 Python 中的列表索引一样，都是从 0 开始计数，例如 userTuple[0] 获取的是元组 userTuple 中的第 1 个元素的值，即 0001。

2. 添加元组

在 Python 中并没有提供向元组中添加元素的函数，比如向列表中添加元素时使用的 append()函数等，但是对元组的添加照样可以实现。因为元组的元素类型可以是字符串、数字，甚至可以是元组，所以可以将一个元组中的元素类型定义为元组，这样就可以实现元组元素的添加了。下面来看一段代码。

```
userTuple = ('0001', '0002', '0003', '0004', '0005', '0006')
# 将 userTuple 元组作为 new_userTuple 元组的元素，并再添加两个元素
new_userTupele=(userTuple, '0007', '0008')
print new_userTupele
```

在该段代码中，首先声明了一个含有 6 个元素的元组，因为元组中的每个元素的类型可以是 Object 类型，因此在 new_userTuple 元组中存放元素时，可以将含有 6 个元素的元组作为元素放入，并向 new_userTuple 元组中添加两个元素，最后使用 print 语句输出新的元组。运行该段代码，输出结果如下：

```
(('0001', '0002', '0003', '0004', '0005', '0006'), '0007', '0008')
```

这样，在 new_userTuple 元组的元素中，含有一个包含 6 个元素的元组和两个字符串类型的元素。遍历该元组时，可以使用嵌套循环将所有元素遍历，从而实现元组元素的添加功能。

5.2.2　基础知识——元组的访问

元组的访问与列表十分相似，同样支持负索引、分片以及多元列表等特性，但是元组中的元素不能修改。

1. 访问普通类型的元组

元组中元素的值可通过索引来访问，访问格式如下：

```
tuple_name [ n ]
```

其中，n 表示要访问元组中元素的索引，索引是从 0 开始的，即要访问元组中的第 3 个元素，索引(下标)为 2。索引 n 的值可以为 0、正整数或负整数(负数索引)。下面演示使用不同的索引来访问元组中元素的方法。

```
userTuple = ('0001', '0002', '0003', '0004', '0005', '0006')
print '元组中的第 3 个元素为：',userTuple[2]
print '元组中倒数第 3 个元素为：',userTuple[-3]
# 使用分片获取第 3 个元素到倒数第 2 个元素组成的元组
print '元组中第 3 个元素到倒数第 2 个元素组成的元组为：',userTuple[2:-2]
```

该段代码演示了使用正整数索引、负数索引和分片索引的方式来访问元组。首先使用正整数索引来访问元组，访问元组中的第 3 个元素，索引为 2；然后使用负数索引来访问元组，访问元组中倒数第 3 个元素，索引为-3，负数索引是从-1 开始的。最后使用分片索引的方式来访问元组，访问元组中第 3 个元素(索引为 2)至倒数第 2 个元素(索引为-2)之间的数据，分片为 2:-2。运行该段代码，输出结果如下：

```
元组中的第 3 个元素为： 0003
元组中倒数第 3 个元素为： 0004
元组中第 3 个元素到倒数第 2 个元素组成的元组为： ('0003', '0004', '0005')
```

创建元组后，其内部元素的值是不能被修改的。要修改元组中的某个元素，运行时将报错。例如：

```
userTuple = ('0001', '0002', '0003', '0004', '0005', '0006')
# 向 userTuple 元组中的第一个元素赋值
userTuple[0]='1'
print userTuple
```

在该段代码中，为 userTuple 元组的第一个元素赋值为 1，运行该段代码，输出错误信息如下：

```
Traceback (most recent call last):
  File "18.py", line 2, in <module>
    userTuple[0]='1'
TypeError: 'tuple' object does not support item assignment
```



元组中的元素不支持赋值操作。

2. 访问二元元组

元组还可以由其他不同的元组组成，即二元元组，二元元组的语法格式如下：

```
tuple_name = ((element1, element2, element3, ...),(element4, element5,
element6, ...))
```

二元元组的创建和访问与二元列表的创建与访问十分相似，下面演示如何创建一个二元元组和访问二元元组中的元素。

```
userTuple1 = ('0001', '0002', '0003')
userTuple2 = ('0004', '0005', '0006')
# 定义二元元组
userTuple=(userTuple1, userTuple2)
print userTuple
# 访问二元元组中第 2 个元组的第 1 个元素
print 'userTuple[1][0]=',userTuple[1][0]
# 访问二元元组中第 2 个元组的第 3 个元素
print 'userTuple[1][2]=',userTuple[1][2]
```

在该段代码中，首先创建一个二元元组，userTuple 由元组 userTuple1 和 userTuple2 组成，即 userTuple 元组为(('0001', '0002', '0003'), ('0004', '0005', '0006'))。接着访问二元元组中第 2 个元素('0004', '0005', '0006')中的第 1 个元素(值为 0004)、第 3 个元素(值为 0006)。运行该段代码，输出结果如下：

```
(('0001', '0002', '0003'), ('0004', '0005', '0006'))
userTuple[1][0]= 0004
userTuple[1][2]= 0006
```

3. 元组的解包操作

在 Python 中，将创建元组的过程称之为打包。相反，元组也可以执行解包操作。解包可以将元组中的各个元素分别赋值给多个变量。这样，避免了使用循环遍历方式来获取每个元素的值，从而降低了代码的复杂性。下面创建一个示例，具体介绍如何为元组打包和解包。

```
userTuple = ('0001', '0002', '0003')
stu1, stu2, stu3 = userTuple
print stu1
print stu2
print stu3
```

首先创建一个 userTuple 元组，元组的创建就是打包的过程，然后使用一个赋值操作，赋值号的右边是 userTuple 元组，因为 userTuple 元组含有 3 个元素，因此在赋值号的左边只能有 3 个变量，这样将 userTuple 元组中的 3 个元素分别赋值给这 3 个变量。运行代码，输出结果如下：

```
0001
0002
0003
```

5.2.3 基础知识——元组的遍历

元组的遍历通常是指通过循环语句 for ... in 依次访问元组中各个元素的值。遍历元组通常需要用到 range()和 len()函数。len()函数不难理解，在前面的案例中，我们已经多次使用到该函数，下面具体介绍一下 range()函数。

range()函数返回一个递增或递减的数字列表，该函数的语法格式如下：

```
range ([start, ] stop [, step])
```

该函数有如下 3 个参数。

- start：该参数是可选的，表示列表开始的值，如果不指定该参数的值，默认为 0。
- stop：该参数是必需的，表示结束的值。
- step：该参数是可选的，表示步长，即每次递增或递减的值，默认值为 1。

1. 使用range()函数实现元组遍历

下面使用 range()函数来遍历一个普通类型的元组，具体了解该函数的使用方法，代码如下：

```
userTuple = ('0001', '0002', '0003', '0004', '0005', '0006')
# 使用 range()函数遍历元组
for item in range(len(userTuple)):
    print userTuple[item]
```

在该段代码中，因为 userTuple 的长度为 6，因此 len(userTuple)的值为 6，即 range(6)的列表为[0, 1, 2, 3, 4, 5]，然后在 for ... in 循环体内输出列表项。运行该段代码，输出结果为：

```
0001
0002
0003
0004
0005
0006
```

二元元组的遍历与二元列表的遍历相同。下面接着演示如何使用 range()函数实现二元元组的遍历，代码如下：

```
userTuple1 = ('0001', '0002', '0003')
userTuple2 = ('0004', '0005', '0006')
# 定义二元元组
userTuple=(userTuple1, userTuple2)
# 使用 range()函数生成列表为[0,1]
for i in range(len(userTuple)):
    # 在此使用 range()函数，生成的列表长度由 userTuple 元组中的元素长度决定
    for j in range(len(userTuple[i])) :
        print 'userTuple['+str(i)+']['+str(j)+']=',userTuple[i][j]
```

该段代码首先定义了一个长度为 2 的二元元组，该二元元组又由两个子元组组成。因为二元元组的长度为 2，因此使用 range()函数生成的列表为[0, 1]，接着使用嵌套循环再次遍历二元元组中的每个元素，即遍历 userTuple 元组中的第 1 个元素时，range(len(userTuple[i]))生成的列表为[0,1,2]。最后使用 print 语句输出遍历结果。运行代码，输出如下：

```
userTuple[0][0]= 0001
userTuple[0][1]= 0002
```

```
userTuple[0][2]= 0003
userTuple[1][0]= 0004
userTuple[1][1]= 0005
userTuple[1][2]= 0006
```

2. 使用map()实现元组遍历

除了可以使用 range()函数实现元组的遍历之外，还可以使用 map()实现对元组的“解包”，得到每个子元组，然后对每个子元组进行遍历，输出 userTuple 元组中所有元素的值。map()函数的语法格式如下：

```
map(function_name, sequence [, sequence, ...])
```

map()函数返回一个自定义函数 function_name 处理后的列表。参数 function_name 为自定义函数名称，该函数用于处理参数 sequence。如果 function_name 的值为 None，则 map()返回一个由参数 sequence 组成的元组或列表。下面来创建一个示例。

```
userTuple1 = ('0001', '0002', '0003')
userTuple2 = ('0004', '0005', '0006')
userTuple=(userTuple1, userTuple2)
for item in map(None, userTuple):
    for i in item:
        print i
```

在该段代码中，首先创建了一个二元元组，这就是“打包”过程，然后使用 map()函数来解包。因为 userTuple 元组中包含两个子元组，而每个子元组都包含 3 个元素，因此使用 map()函数后，userTuple 元组中包含 6 个元素，分别为 0001、0002、0003、0004、0005 和 0006，因此变量 item 在使用 map()函数之后的值变为一个含有 6 个元素的元组——('0001', '0002', '0003', '0004', '0005', '0006')，然后使用嵌套循环遍历 item 元组，将元组中的每一项赋值给变量 i，并输出 i 的值。运行该段代码，输出结果如下：

```
0001
0002
0003
0004
0005
0006
```

5.2.4 实例描述

一个销售水果的商贩，需要清楚地了解每个季节盛产什么种类的水果。我的朋友也不例外，不过他是个“马大哈”，每当换季的时候都不在意这些，就是“随大流”，人家卖什么他卖什么，这样就造成了经济收入不好的状况，做生意要求的就是“与众不同”。自从我给他做了这个水果查询系统之后，他的收入步上一层楼啊，对我真是感激不尽呢！

下面来看看我给他做的水果查询系统吧，其实很简单，但是对外行人士来说就觉得很了不起了。

5.2.5　实例应用

【例 5-2】按季节查询水果系统。

(1) 新建 Python 文件，命名为 season.py。

(2) 在 season.py 文件中编辑代码。首先定义 4 个季节盛产水果的元组，并使用二元元组将不同季节的水果元组存储在一起。代码如下：

```
# 定义春天盛产水果元组
sprints = ('香蕉','杨桃','番荔枝','草莓','柑橘')
# 定义夏天盛产水果的元组
summers=('芒果','黄瓜','番龙眼','西瓜','柠檬')
# 定义秋天盛产水果的元组
autumns=('菠萝','木瓜','杨桃','火龙果','人参果')
# 定义冬天盛产水果的元组
winters=('番石榴','油梨','橙子','苹果')
# 定义二元元组，包含上面定义好的 4 个元组
seasons_fruits=(sprints,summers,autumns,winters)
```

(3) 定义表示季节的元组，代码如下：

```
seasons=('春季','夏季','秋季','冬季')
```

(4) 提示用户输入一个 1～4 之间的数字，分别表示春、夏、秋和冬 4 个季节，然后根据用户输入的数字判断当前季节并输出。代码如下：

```
select_season = raw_input('请选择旅游季节(春天：1，夏天：2，秋天：3，冬天：4)：')
for sea in range(len(seasons)):
    if select_season == str(sea+1):
        print '你选择的是：',seasons[sea]
        print seasons[sea]+'的水果有：'
```

因为在 for ... in 循环中的 sea 所表示的数字是从 0 开始的，而用户输入的数字 select_season 是从 1 开始的，因此需要将 sea+1 与 select_season 进行比较，如果相等，则输出用户选择的季节。

(5) 循环遍历二元元组，根据用户输入的数字选择要遍历的子元组。代码如下：

```
for season in range(len(seasons_fruits)):
    if select_season == str(season+1):
        for fruit in range(len(seasons_fruits[season])):
            print seasons_fruits[season][fruit]
```

在循环体内的 if 条件控制语句中，因为 season 是 int 类型，而用户数组的数字是字符串类型，因此这里需要使用 str()函数来将 int 类型转换为 string 类型，再进行比较。

5.2.6　运行结果

运行 season.py 文件，提示用户“请选择旅游季节”，如图 5-6 所示。当用户输入每个季节的编号时，输出该季节的盛产水果，如图 5-7 所示。

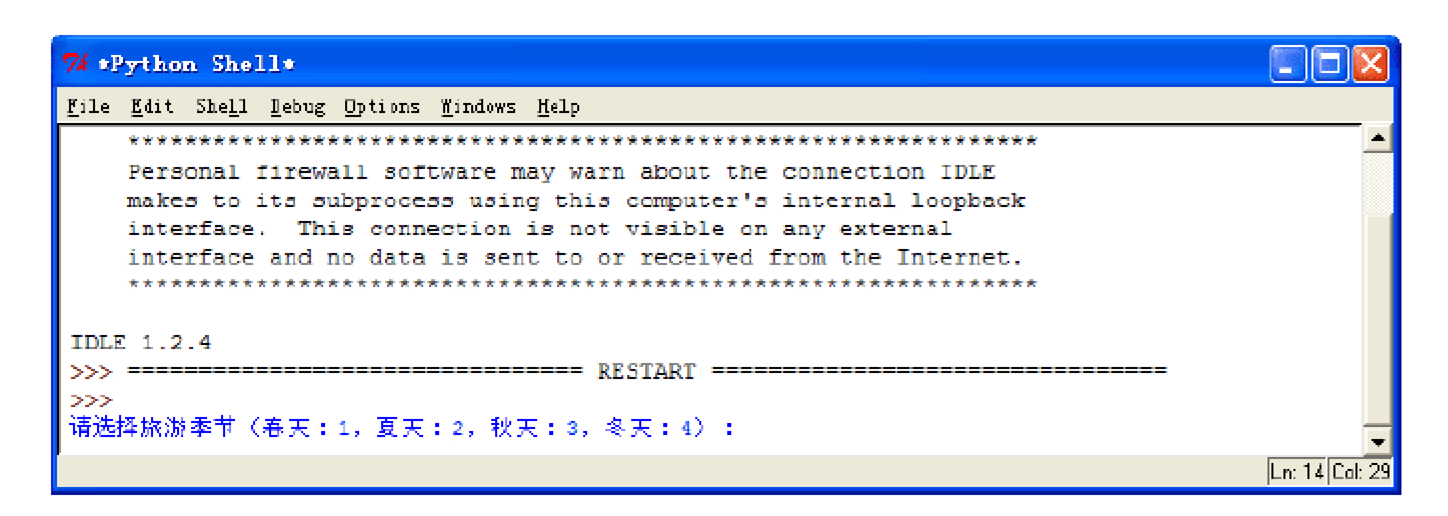


图 5-6　提示用户输入季节编号

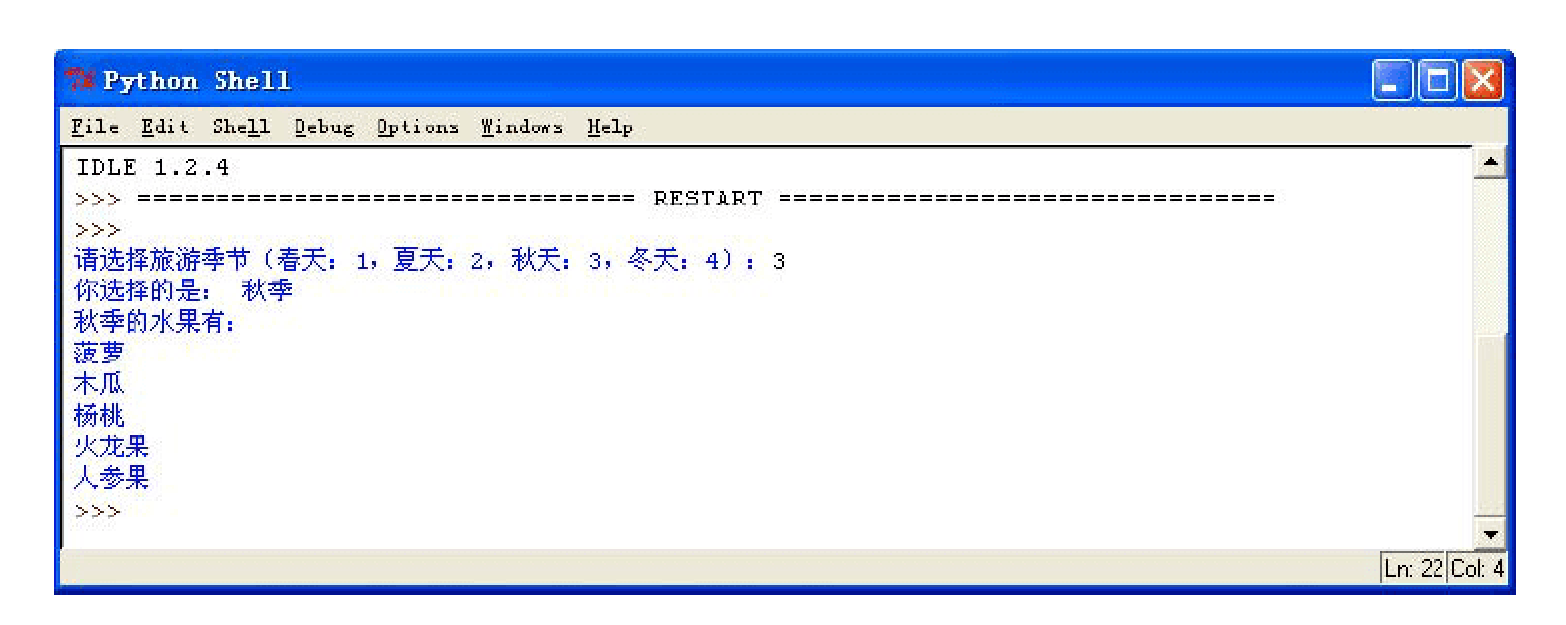


图 5-7　输出所选季节盛产的水果

5.2.7　实例分析

**源码解析**

在上述案例中，首先定义了 4 个元组，并将这 4 个元组作为 seasons_fruits 的子元组，然后使用了 range()和 len()函数循环遍历二次元组，根据用户输入的季节编号将不同季节的水果输出。

5.3　使用字典实现用户账号管理

通过上面的讲解，我们已经了解到：如果需要将值分组到一个结构中，并且通过编号对其进行引用，那么列表(List)就能派上用场了。本节将讲解另一种通过名字引用值的数据结构，这种数据类型称为映射(mapping)，字典是 Python 中唯一内建的映射类型。字典中的值并没有特殊的顺序，但是都存储在一个特定的键(Key)中。键可以是数字、字符串甚至是元组。

视频教学：光盘/videos/05/字典的创建.avi　　**长度：**6 分钟

视频教学：光盘/videos/05/字典的方法.avi　　**长度：**13 分钟

视频教学：光盘/videos/05/字典的基本操作.avi　　**长度：**15 分钟

5.3.1 基础知识——字典的创建

字典由一系列的“键-值”(key-value)对组成，键-值对之间用逗号隔开，并且包含在一对花括号中“{}”，没有顺序，如果需要特定的顺序，那么应该在使用前对这些键-值对进行排序。字典与 Java 语言中的 HashMap 类的作用类似，都是采用键-值对映射的方式存储数据。

1. 基本字典的创建

字典的创建非常简单，创建字典的语法格式如下：

```
dictionary_name = {key1 : value1, key2 : value2, key3:value3,...}
```

其中，key1、key2、key3 等表示字典的键值(key)，valuex 表示字典的 value 值。字典由多个键及与其对应的值成对组成，每个键及其对应的值之间用冒号隔开，项与项之间用逗号隔开，而整个字典由一对大花括号括起来。

空字典(不包括任何项)由两个大花括号组成，语法如下：

```
dictionary_name = {}
```



字典中的键是唯一的(其他类型的映射也是如此)，而值并不唯一。

下面创建一个示例，演示字典的创建方法。代码如下：

```
userDic = {'0003':'June','0002':'Tom'}
print userDic
```

在该示例中，使用学生编号引用学生姓名。运行该段代码，输出结果如下：

```
{'0002': 'Tom', '0003': 'June'}
```

从运行结果来看，代码中书写的顺序并不是字典中的实际存储顺序，字典将根据每个元素的 Hashcode 值进行排列。在创建字典时，也可以使用数字作为索引。

```
userDic = {3:'June',2:'Tom'}
```

2. 使用dict()函数创建字典

前面讲解了如何创建一个用花括号括起来的字典类型。其实，通过其他映射或者类似于(key,value)的序列对也可以创建字典。这里需要用到 dict()函数，下面来看一段代码。

```
userDic = [(2,'maxianglin'),(1,'wanglili'),(3,'malinlin')]
dic_userDic = dict(userDic)
print dic_userDic
```

在该段代码中，首先定义了一个列表，列表中有 3 个元素，每个元素都是一个字典，然后使用 dict()函数将 userDic 列表转换成字典类型，最后输出结果为：

```
{1: 'wanglili', 2: 'maxianglin', 3: 'malinlin'}
```

dict()函数也可以通过关键字参数来创建字典，代码如下：

```
userDic = dict(name='maxianglin',age=24,sex='0')
print userDic
```

在 dict()函数中使用变量的形式来存储键-值对。运行代码，输出结果如下：

```
{'age': 24, 'name': 'maxianglin', 'sex': '0'}
```

5.3.2 基础知识——字典的基本操作

字典与列表一样，可以进行增加元素、删除元素、访问元素和遍历字典元素等操作。下面详细介绍一下字典的基本操作。

1. 向字典中添加元素

虽然字典与列表有很多相似之处，但是字典是无序的，因此字典中没有 append()方法，如果需要向字典中插入一个新的元素，则可以调用 setdefault()方法。该方法可以创建一个新的元素并设置默认值。setdefault()方法的声明如下：

```
setdefault (key [, default_value])
```

其中，参数 key 表示字典的键值，参数 default_value 表示字典元素的默认 value 值。参数 default_value 为可选参数，如果不指定该参数的值，默认为 None。要添加的参数 key 的值如果已经在字典中存在，那么 setdefault()函数将返回原有的值；参数 key 的值如果在字典中不存在，则字典将被添加一个新的元素，并返回新元素的 value 值。下面创建一个示例，演示如何使用 setdefault()方法向字典中添加新元素。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
# 添加元素，键为 0004，值为 zhuhongtao
userDic.setdefault('0004','zhuhongtao')
# 输出添加元素后的值
print userDic
# 向列表中添加值，但是参数 0001 在 userDic 字典中已经存在
userDic.setdefault('0001','zhangfang')
print userDic
```

在该段代码中，首先创建一个含有 4 个元素的字典，然后使用 setdefault()函数向列表中添加一个新的元素，该新元素的 key 为 0004，value 为 zhuhongtao，最后输出添加元素后的字典。

```
{'0004': 'zhuhongtao', '0001': 'maxianglin', '0002': 'wanglili', '0003':
'malinlin'}
```

接着使用 setdefault()函数向列表中添加元素，但是 key 为 0001 的键已经在 userDic 字典中存在，因此返回的是原有的值 maxianglin，而非 zhangfang，输出结果如下：

```
{'0004': 'zhuhongtao', '0001': 'maxianglin', '0002': 'wanglili', '0003':
'malinlin'}
```

上面讲解的是通过 setdefault()函数向字典中添加一个元素。一般情况下，不用此函数同样可以向字典中添加元素，只需要一条赋值语句即可。

```
dictionary_name['key'] = 'value'
```

如果这条语句中的 key 不在字典 dictionary_nam 的 key 列表中，那么字典 dictionary_nam 将被添加一条新的映射(key:value)；如果键 key 已在字典 dictionary_nam 中，那么字典

dictionary_nam 将直接修改 key 对应的 value 值。下面来看一段代码。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
# 添加一个新的元素
userDic['0004']='zhangfang'
print userDic
# 修改键为 0002 的值
userDic['0002']='yangrui'
print userDic
```

在该段代码中，因为 userDic 字典中并没有 0004 的键，因此 userDic['0004']='zhangfang'向字典中添加了一个新元素，该元素的键为 0004，值为 zhangfang。下面使用 userDic['0002']='yangrui'将字典中键为 0002 的值修改为 yangrui，并没有再添加元素。运行代码，输出结果如下：

```
{'0004': 'zhangfang', '0001': 'maxianglin', '0002': 'wanglili', '0003':
'malinlin'}
{'0004': 'zhangfang', '0001': 'maxianglin', '0002': 'yangrui', '0003':
'malinlin'}
```

2. 删除字典中原有的元素

字典与列表不同，字典中没有 remove()函数。但是字典的删除有多种方法实现，下面作详细介绍。

1) 使用 del()方法删除字典中的元素

对字典元素的删除，可以调用 del()来实现。del()属于内建函数，不需要使用 import 语句导入，直接调用即可，其语法格式如下：

```
del (dictionary_name[key])
```

其中，key 表示在 dictionary_name 字典中已经存在的键，即需要删除元素所对应的键。下面来创建一个示例。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
del(userDic['0002'])
print userDic
```

在该段代码中，首先创建了一个含有 3 个元素的字典，然后使用 del()函数将键为 0002 的元素删除，最后输出删除后字典的值。运行代码，输出结果如下：

```
{'0001': 'maxianglin', '0003': 'malinlin'}
```

2) 使用 pop()方法删除字典中的元素

可以调用 pop()方法来弹出列表中的一个元素，字典也有一个 pop()方法，该方法的声明和作用与列表的 pop()不同，用于删除并返回指定键的条目。字典的 pop()方法的声明如下：

```
pop(key [, default_value])
```

pop()方法必须指定参数 key，才能删除对应的值。如果字典中存在 key，则返回值为 key 所对应的值，否则返回 default_value。下面创建一个示例。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
print userDic.pop('0002')
print userDic
```

该段代码首先创建了一个含有 3 个元素的字典，然后使用 pop()方法删除键为 0002 所对应的值，并返回键 0002 所对应的值，最后输出删除后的 userDic 字典。输出结果如下：

```
wanglili
{'0001': 'maxianglin', '0003': 'malinlin'}
```

3) 使用 del 保留字删除字典中的元素

删除字典中的元素除了可以使用 del()方法之外，还可以使用 del 保留字，语法格式如下：

```
del dictionary_name[key]
```

与 del()函数颇有相似之处，都需要指定要删除的 key。下面来举个例子：

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
del userDic['0002']
print userDic
```

运行该段代码，输出结果与使用 del()方法删除字典中元素的结果是相同的。

3. 字典的遍历

字典的遍历有很多种方法，与其他数据结构相同，使用 for ... in 循环语句可以完成字典的遍历。遍历字典时，需要不断地获取字典中的元素，这就需要访问字典。字典的访问与元组、列表有所不同，元组和列表是通过数字索引来获取对应的值，而字典是通过 key 值来获取对应的 value 值。

访问字典是通过 key 值来获取所对应的 value 值，语法格式如下：

```
value = dictionary [key]
```

下面创建一个小程序，访问字典中的元素。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
print userDic['0003']
```

该段代码访问了 userDic 字典中键为 0003 所对应的值，最后输出结果为：

```
malinlin
```

遍历字典最直接的方法是通过 for ...in 循环语句来完成，当然在 Python 中还提供了多个方法用于遍历字典。

1) 使用 for ...in 循环遍历字典

与 Python 中的其他数据结构相同，可以采用 for ...in 循环遍历字典。下面这段代码演示了字典的遍历操作。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
for key in userDic:
    print 'userDic[%s]='% key,userDic[key]
```

在该段代码中，使用了 for ...in 循环遍历 userDic 字典。需要注意的是，在循环体内的输出语句中，%s 表示要输出 userDic 字典的内容，%key 表示%s 所对应的值，即 userDic 字典中的所有 key。运行代码，输出结果如下：

```
userDic[0001]= maxianglin
userDic[0002]= wanglili
userDic[0003]= malinlin
```

2) 使用 items()方法遍历字典

在 Python 中，提供了一个专门用来遍历字典的方法，该方法的使用如下：

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
print userDic.items()
```

运行该段程序，输出结果如下：

```
[('0001', 'maxianglin'), ('0002', 'wanglili'), ('0003', 'malinlin')]
```

可见，item()方法把字典中每对 key 和 value 组成一个元组，并把这些元组存放在列表中返回。那么，如何输出使用 items()方法之后的列表中的元素呢？下面再来看一段代码。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
for (key, value) in userDic.items():
   print 'userDic[%s]='% key, value
```

因为使用 items()方法之后获取的结果为[(key1,value1),(key2,value2)]序列，所以在遍历 userDic.items()时，声明变量(key,value)将每个元素的键和值分别赋值给 key 和 value，这样 key 所存储的就是 userDic 字典中的键，value 所存储的就是 userDic 字典中键所对应的值。最后将遍历后的结果输出。运行该段代码，输出结果如下：

```
userDic[0001]= maxianglin
userDic[0002]= wanglili
userDic[0003]= malinlin
```

3) 使用 iteritems()、iterkeys()和 itervalues()方法遍历字典

字典的遍历还可以使用 iteritems()、iterkeys()和 itervalues()方法来实现，这些方法返回一个遍历器对象，通过这个对象可以实现遍历的输出。下面先使用 iteritems()方法做一个示例。

```
userDic={'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
for (key,value) in userDic.iteritems():
   print 'userDic[%s]='% key,value
```

iteritems()与 items()方法有点相似，返回的都是(key,value)序列。运行代码，输出结果如下：

```
userDic[0001]= maxianglin
userDic[0002]= wanglili
userDic[0003]= malinlin
```

下面再使用 iterkeys()和 itervalues()方法来创建一个示例，代码如下：

```
for key in userDic.iterkeys():
   print key
for value in userDic.itervalues():
   print value
for (key, value) in zip(userDic.iterkeys(),userDic.itervalues()):
   print 'userDic[%s]='% key,value
```

该段代码中的第 1 个循环遍历 userDic 字典中的所有 key，第 2 个循环遍历 userDic 字典中的所有 value，第 3 个循环遍历 userDic 字典中的所有元素。使用 zip()函数将 userDic.iterkeys(),userDic.itervalues()作为一个整体，把 userDic.iterkeys()的值赋给 key，把 userDic.itervalues()的值赋给 value，最后输出。运行代码，输出结果如下：

```
0001
0002
```

```
0003
maxianglin
wanglili
malinlin
userDic[0001]= maxianglin
userDic[0002]= wanglili
userDic[0003]= malinlin
```

5.3.3 基础知识——字典的方法

就像其他内建类型一样，字典有很多重要的方法。下面讲述几个在程序开发过程中经常用到的方法。

1. clear()方法

clear()方法用于清除字典中所有的项，无返回值(或者说返回值是 None)。该方法的语法格式如下：

```
dictionary_name.clear()
```

下面使用该方法创建一个示例。

```
userDic={'name':'maxianglin','age':23,'sex':'female'}
print '调用 clear()方法之前：',userDic
# 调用 clear()方法，清除 userDic 字典中的所有元素
userDic.clear()
print '调用 clear()方法之后：',userDic
```

运行该段代码，输出结果如下：

```
调用 clear()方法之前 {'age': 23, 'name': 'maxianglin', 'sex': 'female'}
调用 clear()方法之后 {}
```

2. copy()方法

copy()方法返回一个具有相同键-值对的新字典，该方法的语法格式如下：

```
dictionary_targetName=dictionary_sourceName.copy()
```

其中，dictionary_targetName 表示要复制的目标字典名称，dictionary_sourceName 表示源字典名称。下面创建一个示例，具体了解如何使用 copy()方法将原有的字典复制给一个新字典。

```
source_userDic= {'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
# 调用 copy()方法，将 source_userDic 字典复制于 target_userDic 字典，形成一个具有相同
# 键-值对的新字典
target_userDic=source_userDic.copy()
print '修改之前的目标字典 target_userDic=',target_userDic
# 修改目标字典键为 0002 的值为 zhangfang
target_userDic['0002'] = 'zhangfang'
# 输出源字典中的元素
print '源字典 source_userDic=',source_userDic
print '修改之后的目标字典 target_userDic=',target_userDic
```

该段代码首先使用 copy()方法将源字典中的内容复制到目标字典中，形成一个具有相同键

-值对的新字典 target_userDic，然后修改新字典中键为 0002 的值，将修改后的源字典 source_userDic 与新字典 target_userDic 的内容输出。运行该段代码，输出结果如下：

```
修改之前的目标字典 target_userDic= {'0001': 'maxianglin', '0002': 'wanglili',
'0003': 'malinlin'}
原字典 source_userDic= {'0001': 'maxianglin', '0002': 'wanglili', '0003':
'malinlin'}
修改之后的目标字典 target_userDic= {'0001': 'maxianglin', '0002': 'zhangfang',
'0003': 'malinlin'}
```

从输出结果可以看出，当在副本中替换值的时候，原始字典不受任何影响。但是，如果修改了原始字典中的某个值，副本字典是会改变的。

3．fromkeys()方法

fromkeys()方法使用给定的键来建立新的字典，每个键默认对应的值为 None。该方法的语法格式如下：

```
dirctionary_name.fromkeys([key1, key2, ...], (default_value))
```

该方法有两个参数：第一个参数的类型为列表，在该列表中定义了要创建的新字典中的所有键(key)，该列表是由字典中的所有键组成的；第二个参数是可选的，该参数指定了新字典中所有键所对应的默认值。在这里，只能指定一个值，即所有键的值是相同的，如果没有指定该参数的值，则默认为 None。

```
print {}.fromkeys(['0001','0002'])
```

在该段代码中，首先构造了一个空的字典，然后调用它的 fromkeys()方法，建立另外一个字典，输出结果为：

```
{'0001': None, '0002': None}
```

其实也可以直接在一个非空字典中调用 fromkeys()方法，例如：

```
source_userDic= {'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
print source_userDic.fromkeys(['0001','0002'])
```

输出结果和使用空字典的 fromkeys()方法是相同的。如果需要使用其他值作为默认值，而非 None 值，则可以自定义默认值，例如：

```
print source_userDic.fromkeys(['0001','0002'],'maxianglin')
```

输出结果如下：

```
{'0001': 'maxianglin', '0002': 'maxianglin'}
```

4．get()方法

get()方法用于访问字典中的某个元素，返回该元素的 value 值。如果访问的字典元素不存在，则返回 None。该方法的语法格式如下：

```
dictionary_name.get(key)
```

其中，key 表示要访问的字典中的键。如果使用 dictionary_name[key]的形式访问字典中的元素，当访问的字典元素不存在时，则会报告以下错误：

```
Traceback (most recent call last):
  File "34.py", line 5, in <module>
    source_userDic['0004']
KeyError: 'xxx'
```

而使用 get()方法访问正好解决了这个问题，它返回的是 None。下面来看一段代码。

```
userDic= {'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
print userDic.get('0002')
print userDic.get('0004')
print userDic['0004']
```

在代码的第 2 行，使用 get()方法访问字典中键为 0002 的值，返回所对应的 wanglili；在代码的第 3 行，使用 get()方法访问字典中键为 0004 的值，因为该键不存在，所以返回的是 None；在代码的第 4 行，使用 dictionary_name[key]的形式访问字典中键为 0004 的值，该键在字典中不存在，因此报出错误。运行该段代码，输出结果如下：

```
wanglili
None
Traceback (most recent call last):
  File "35.py", line 4, in <module>
    print userDic['0004']
KeyError: '0004'
```



从上面的示例可以看出，当使用 get()方法访问一个不存在的键时，不会出现任何异常，而得到的是 None 值。

使用 get()方法不仅可以实现访问不存在的元素能正常运行之外，还可以自定义默认值，将 None 值替换掉，请看下面的例子。

```
print userDic.get('0004','zhanghui')
```

该条语句输出的结果为 zhanghui，从而代替了原来的 None 值。

5. has_key()方法

has_key()方法可以检查字典中是否含有指定的键，如果含有，返回 Ture，否则返回 False。该方法的语法格式如下：

```
dictionary_name.has_key(key)
```

该方法的使用非常简单。下面创建一个示例，具体介绍该方法的使用。

```
userDic= {'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
print userDic.has_key('0004')
print userDic.has_key('0002')
```

运行该段代码，输出的结果如下：

```
False
True
```

6. popitem()方法

popitem()方法类似于列表中的 pop()方法，后者会弹出列表的最后一个元素。不同的是，popitem()方法弹出随机的元素，并非字典中的最后一个元素，因为在字典中并没有“最后的元素”或者其他有关顺序的概念。该方法的语法格式如下：

```
dictionary_name.popitem()
```

下面创建一个示例，具体介绍该方法的使用。

```
userDic= {'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
# 第一次调用 popitem()方法
print userDic.popitem()
# 第二次调用 popitem()方法
print userDic.popitem()
print userDic
```

在该段代码中，首先调用了字典的 popitem()方法，弹出字典中的第一个元素项，然后再次调用 popitem()方法，程序会弹出字典中的第二个元素项，依次类推。当第 N 次调用 popitem()方法之后，将弹出字典中的第 N 个元素项。如果调用的次数比字典中的元素个数多，则抛出 KeyError: 'popitem(): dictionary is empty'异常。运行该段代码，输出结果如下：

```
('0001', 'maxianglin')
('0002', 'wanglili')
{'0003': 'malinlin'}
```

第一次调用 popitem()方法，弹出 userDic 字典中的第一项('0001', 'maxianglin')；第二次调用时，弹出第二项('0002', 'wanglili')。最后将 userDic 字典中的元素输出，即输出除了弹出元素项之外的元素，故只剩下键为 0003 的元素值。

7．update()方法

update()方法可以利用一个字典项更新另外一个字典，无返回值。该方法的语法格式如下：

```
update_dictionary.update(source_dictionary)
```

其中，update_dictionary 表示新的字典，该字典中一般有一个元素；source_dictionary 表示要更改的字典。下面创建一个示例，具体介绍如何使用 update()方法来更新字典中的元素项。

```
userDic= {'0001':'maxianglin','0002':'wanglili','0003':'malinlin'}
newDic={'0002':'zhangfang'}
# 使用 update()方法，更新 userDic 字典中键为“0002”的元素
userDic.update(newDic)
print userDic
```

在该段代码中定义的 newDic 中的键 0002 已经在 userDic 字典中存在，因此 newDic 字典中的元素项会将原来字典 userDic 中的元素项覆盖。输出结果如下：

```
{'0001': 'maxianglin', '0002': 'zhangfang', '0003': 'malinlin'}
```

如果要更改的元素项在原来字典中不存在，则将提供的字典中的元素项添加到 userDic 字典中。将上段代码中的 newDic 中元素项的键由 0002 更改为 0004，再次运行代码，输出结果如下：

```
{'0004': 'zhangfang', '0001': 'maxianglin', '0002': 'wanglili', '0003':
'malinlin'}
```

5.3.4　实例描述

在开发过程中，往往需要首先注册账户，然后才可以登录到程序的主页面。下面就来做一个类似于用户账号管理的程序，程序运行时，可以选择注册账户，或者登录已有账户，登录成功则返回欢迎信息，否则需要用户重新登录。

5.3.5　实例应用

【例 5-3】使用字典实现用户账号管理。

(1) 新建 Python 文件，命名为 login.py。

(2) 在该文件中编辑代码，首先定义一个空的字典，用于存储用户注册的用户名和密码。代码如下：

```
db={}    #声明一个空的字典
```

(3) 编辑注册账户的函数，命名为 newuser()，在该函数中首先使用 raw_input()函数让用户输入要注册的账号，然后使用 has_key()函数检测该账号是否已经在 db 字典中存在，如果存在，重新输入，否则让用户输入注册密码，该密码作为用户输入账号的 value 值，账号作为 key，存储于 db 字典中。代码如下：

```
# 如果是新用户，则需要注册
def newuser ():
    prompt='请输入注册账号：'
    while True:
        name=raw_input(prompt)
        # 检测字典中是否已经存在键为用户注册账号的元素
        if db.has_key(name):
            prompt='您注册的账号已经存在，请使用其他账号注册：'
            continue
        else:
            password=raw_input('请输入注册密码：')
            # 将用户注册的账号和密码作为字典的键-值对
            db[name]=password
            break
```

(4) 编辑用户登录函数，命名为 olduser()。如果用户已注册过，则执行该函数。在该函数中首先使用 raw_input()函数，让用户输入登录账号和密码；然后使用字典的 get()方法，根据用户输入的登录账号获取 db 字典中对应的登录密码；再使用 if 条件控制语句，判断用户输入的密码是否正确，如果正确，提示欢迎信息，否则重新登录。olduser()函数的定义如下：

```
# 如果已经注册过，则需要登录
def olduser ():
    name=raw_input('请输入登录账号:')
    password=raw_input('请输入登录密码:')
    # 获取注册账号所对应的密码
    userpwd=db.get(name)
    # 判断用户输入的登录密码是否与注册密码相同
    if userpwd == password:
```

```
        print '欢迎您登录：',name
    else:
        print '您的用户名或密码错误，请重新登录！'
```

(5) 创建显示界面函数 showmenu()。在该函数中，首先让用户输入账号状态，n 表示注册，新用户；e 表示登录，已经注册过的用户。如果用户输入的是 n，则执行 newuser()函数；如果是 e，则执行 olduser()函数，否则让用户重新输入状态。showmenu()函数的定义如下：

```
# 显示系统界面
def showmenu ():
    prompt="请输入用户状态(n：注册 e：登录)："
    con = False
    while not con:
        chosen = False
        while not chosen:
            try:
                # 将用户输入的字母小写格式化
                choice=raw_input(prompt).strip()[0].lower()
            except(EOFError,KeyboardInterrupt):
                choice='q'
            print '您按下了【%s】键'% choice
            if choice not in 'neq':
                print '您输入的内容不合法，请重新输入：'
            else:
                chosen = True
                con = True
        if choice == 'n':
            newuser()
        elif choice == 'e':
            olduser()
        else:
            showmenu()
        showmenu()
```

(6) 最后调用 showmenu()函数，即在 login.py 文件的结尾输入：

```
showmenu()
```

5.3.6　运行结果

运行 login.py 文件，系统提示用户输入用户状态，如果输入的既不是 n，也不是 e，则提示用户重新输入。当用户输入的密码并非与注册密码相同时，则提示“密码错误”的信息，并提示用户重新登录，如图 5-8 所示。

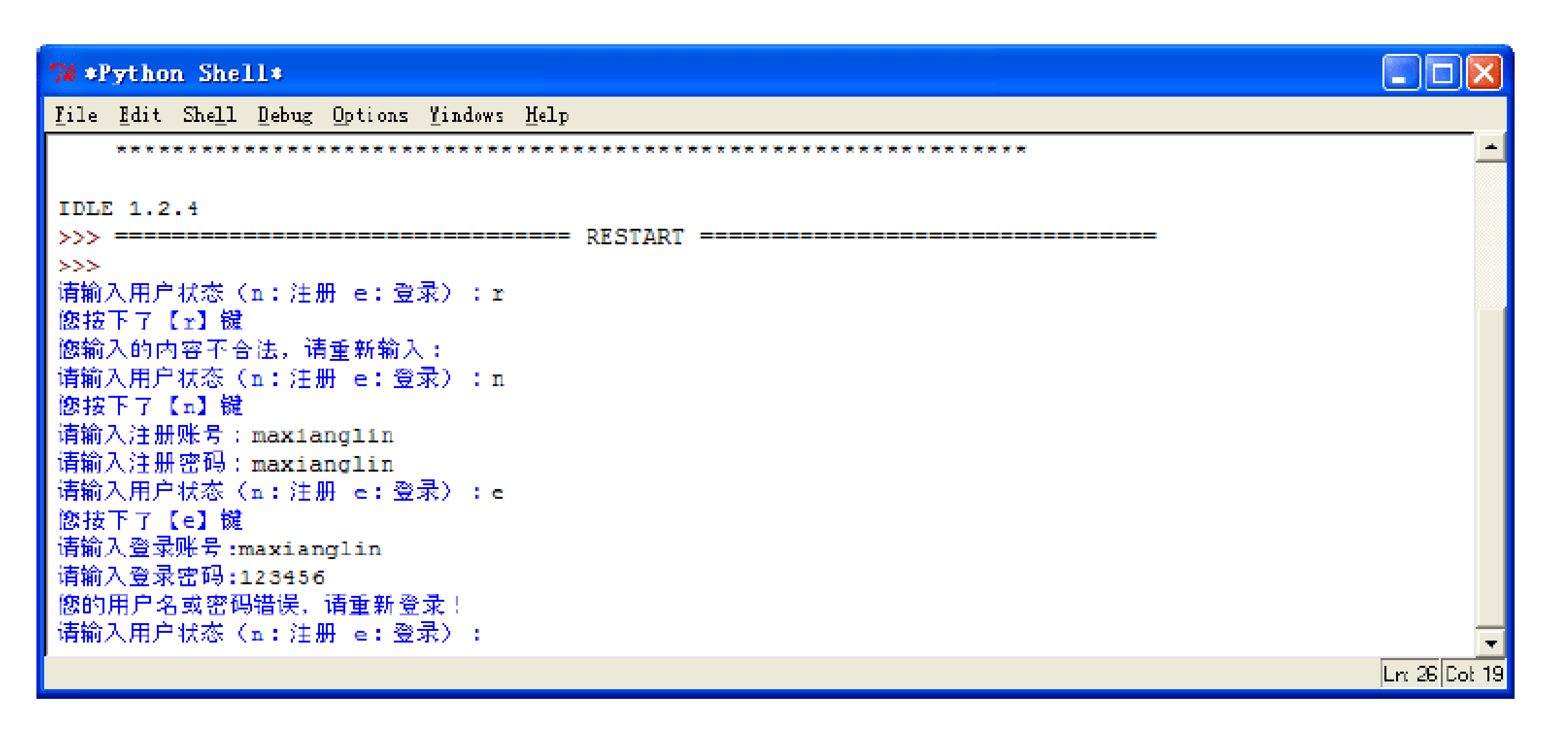


图 5-8　登录密码错误

当用户输入的密码与注册密码相同时，则提示“欢迎”信息，如图 5-9 所示。

```
*Python Shell*
File Edit Shell Debug Options Windows Help
IDLE 1.2.4
>>> ================================ RESTART ================================
>>>
请输入用户状态（n：注册 e：登录）：n
您按下了【n】键
请输入注册账号：maxianglin
请输入注册密码：maxianglin
请输入用户状态（n：注册 e：登录）：e
您按下了【e】键
请输入登录账号:maxianglin
请输入登录密码:maxianglin
欢迎您登录： maxianglin
请输入用户状态（n：注册 e：登录）：
Ln: 23 Col: 19
```

图 5-9　登录成功

5.3.7　实例分析



源码解析

在上述案例中，使用了一个名称为 db 的空字典来控制用户的账号和密码，该字典中只有一个元素，即用户输入的注册账号作为元素的键(key)，用户输入的注册密码作为元素的值(value)。在登录函数中，使用了 db.get(name)方式来获取字典中存放的注册密码，接着使用用户输入的密码与注册密码作比较，检测是否相同，如果相同则登录成功，提示欢迎信息。

5.4　序　　列

列表、元组和字符串都是序列，序列的两个主要特点是索引操作符和切片操作符。索引操作符可以从序列中抓取一个特定的元素项，切片操作符可以获取序列的一个切片。本节将详细介绍序列中的某些特定操作。

 **视频教学**：光盘/videos/05/序列.avi　　　　长度：8 分钟

5.4.1　基础知识——序列的索引

序列中的所有元素都具有编号，索引从 0 开始，这些元素可以通过索引来访问。下面创建一个示例，具体介绍序列中通过索引访问数据的方法。

```
userStr='maxianglin'
userTuple=('maxianglin','wanglili','wangjunchao')
userList=['maxianglin','wanglili','wangjunchao']
print userStr[0]
print userStr[-2]
print userTuple[-2]
print userList[2]
```

在代码的第 4 行和第 5 行分别访问了字符串 userStr 中的第 1 个字符和倒数第 2 个字符，输出结果为：

```
m
i
```

在代码的第 7 行，访问了元组中的倒数第 2 个元素，输出结果为：

```
wanglili
```

在代码的第 8 行，访问了列表中的第 3 个元素，输出结果为：

```
wangjunchao
```

5.4.2 基础知识——序列的分片

与使用索引来访问单个元素类似，可以使用分片操作来访问一定范围内的元素，分片通过冒号相隔的两个索引来实现。分片操作对提起序列中的部分数据很有用，而索引在这里显得尤为重要。第 1 个索引是需要提取部分的第 1 个元素的索引，而最后的索引则是分片之后剩下部分的第 1 个元素的索引。代码如下：

```
numbers= [0,1,2,3,4,5,6,7,8,9]
print numbers[3:6]
```

该段代码获取了 numbers 列表中的第 4 个元素到第 7 个元素之间的部分数据，其中包括第 4 个元素而不包括第 7 个元素。第 4 个元素的值为 3，第 7 个元素的值为 6，即获取的是 3、4 和 5。运行该段代码，输出结果为：

```
[3, 4, 5]
```

1. 优雅的捷径

如果需要访问序列的最后 3 个元素，不需要同时指定分片的起始位置和结束位置，只需要指定分片的起始位置即可。代码如下：

```
numbers= [0,1,2,3,4,5,6,7,8,9]
print numbers[7:]
```

该段代码表示获取 numbers 列表中索引为 7 之后的元素，索引为 7 的元素值为 7，因此使用 7:获取了 numbers 列表中的最后 3 个元素。输出结果如下：

```
[7, 8, 9]
```

如果需要从列表的末尾开始计数亦是如此，代码如下：

```
numbers= [0,1,2,3,4,5,6,7,8,9]
print numbers[-3:]
```

该段代码表示获取从末尾开始反向计数的最后 3 个元素，输出结果和上面代码相同。

2. 更大的步长

进行分片的时候，分片的开始和结束点需要进行指定(间接或直接)，而另外一个参数——步长通常都是隐式设置的。在普通的分片中，步长是 1，分片操作就是按照这个步长逐个遍历序列的元素，然后返回开始和结束点之间的所有元素。下面使用分片处理的步长参数创建一个示例，代码如下：

```
numbers= [0,1,2,3,4,5,6,7,8,9]
print numbers[0:10:1]
```

在该段代码中，显示地指定了步长为 1，默认也是 1。执行该段代码，输出结果如下：

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

如果步长被设置为比 1 大的数，那么就会跳过某些元素。下面再来看一段代码。

```
numbers= [0,1,2,3,4,5,6,7,8,9]
```

```
print numbers[0:10:2]
```

输出结果为：

```
[0, 2, 4, 6, 8]
```

在这两个示例中，都指定了分片处理的起始位置和结束位置。其实，之前提到过的捷径也可以使用，比如需要将每 3 个元素中的第 1 个元素提取出来，那么只要设置步长为 3 即可。

```
numbers= [0,1,2,3,4,5,6,7,8,9]
print numbers[::3]
```

执行该段代码，输出结果如下：

```
[0, 3, 6, 9]
```

5.4.3　基础知识——序列相连

通过使用加号，可以进行序列的连接操作。

```
numbers=[0,1,2,3]+[4,5,6,7]
print numbers
```

该段代码将两个列表相连，执行代码，输出如下：

```
[0, 1, 2, 3, 4, 5, 6, 7]
```

除了列表可以相连之外，元组和字符串也可以使用+号实现相连。



列表与列表、元组与元组、字符串与字符串之间都可以使用+号实现相连，但是不同类型的序列不能进行相连操作，比如字符串和列表。

5.4.4　基础知识——序列的乘法

在数字运算中，当一个数 x 乘以另一个数 y 时，y 被扩大 x 倍，而在一个序列中，当一个数 x 乘以一个序列时，该序列将被重复 x 次，从而生成一个新的序列。下面创建一个示例，具体介绍序列中的乘法。

```
userIdStr='0001'*6
print userIdStr
userIdList=['0001']*5
print userIdList
```

该段代码的第 1 行使用字符串 0001 与数 6 相乘，userIdStr 字符串变为：

```
000100010001000100010001
```

该段代码的第 3 行使用列表['0001']与数 5 相乘，userIdList 列表变为：

```
['0001', '0001', '0001', '0001', '0001']
```

5.5 常见问题解答

5.5.1 检测列表中的元素



如何检测某个元素是否在列表中？

网络课堂：http://bbs.itzcn.com/thread-15815-1-1.html

最近在学习 Python 的数据结构，定义了一个名称为 list 的列表，我需要检测在 list 列表中某个元素是否存在，Python 有没有可以实现这种功能的方法呢？我的代码如下：

```
list=['apple','banana','orange','tomato']
fruit='orange'
```

有没有一个函数能直接用来检测 fruit 是否在 list 中呢？

【解决办法】使用 if 条件控制语句可解决该问题。代码如下：

```
fruitList=['apple','banana','orange','tomato']
fruit='orange'
if fruit in fruitList:
    print 'orange 已经存在'
```

无论是定义变量还是定义数据结构名称，不要使用 list 作为名称，这是系统自带的函数，容易造成不必要的麻烦。

5.5.2 Python字典排序问题



Python 中字典的排序问题？

网络课堂：http://bbs.itzcn.com/thread-15816-1-1.html

现在有一个字典，里面的 key 是一篇文章的单词，value 是这个单词在文档中出现的次数，如果我想做个排序，按 key 或按 value 来排列应该怎么做呢？

【解决办法】字典的排序可以使用 sorted()方法实现。代码如下：

```
worldDic={'a':9, 'world':10, 'z':8,'hello':12}
print sorted(worldDic.items(),key=lambda d:d[0])
print sorted(worldDic.items(),key=lambda d:d[1])
```

在第 2 行代码中输出按照 worldDic 中的 key 排序后的字典内容，第 3 行代码输出按照 worldDic 中的 value 排序后的字典内容。运行该段代码，输出结果如下：

```
[('a', 9), ('hello', 12), ('world', 10), ('z', 8)]
[('z', 8), ('a', 9), ('world', 10), ('hello', 12)]
```

5.6　习　　题

一、填空题

(1) 设 s='abcdefg'，则 s[3]值是__________，s[3:5]值是__________，s[:5]值是__________，s[-2:-5]值是__________。

(2) 删除字典中的所有元素的函数是__________，返回包含字典中所有键的列表的函数是__________，返回包含字典中所有值的列表的函数是__________。

(3) 定义一个名称为 fruitList 的列表，如下所示：

```
fruitList=['apple','banana','orange','tomato']
print fruitList[__________]
```

如果输出的结果为 orange，则画线处应填写__________。

二、选择题

(1) 以下不能创建一个字典的语句是__________。

A. dict1 = {}　　B. dict2 = {3:5}

C. dict3 = dict([2,5],[3,4])　　D. dict4 = dict(([1,2],[3,4]))

(2) 判断一个键在字典中是否存在的函数是__________。

A. has_key()　　B. keys()　　C. key()　　D. items()

(3) 下面代码的输出结果为__________。

```
fruit1=('apple','banana','orange')
fruit2=('tomato','pear','berries')
fruits=(fruit1,fruit2)
print fruits[1][1]
```

A. banana　　B. apple　　C. tomato　　D. pear

三、上机练习

上机练习： 实现根据用户名查询联系方式的功能。

编写 Python 代码，实现当用户输入用户名时，检测该用户名是否在已有的字典中存在，如果已经存在，则根据用户输入的不同内容查询不同的联系方式(p 表示用户要查询的是该用户名所对应的联系电话，a 表示用户要查询的是该用户名所对应的家庭住址)，如图 5-10 所示。如果用户输入的用户名在已经定义的字典中不存在，则退出系统，重新输入用户名，如图 5-11 所示。

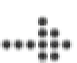




图 5-10 输入的用户名在字典中已经存在



图 5-11 输入的用户名在字典中不存在



第 6 章　字符串与正则表达式

内容摘要

字符串的概念想必大家对它已经再熟悉不过了，学过 C、Java、PHP 等语言的读者都非常了解字符串的概念，因为在任何一种编程语言和程序中都会有它的身影，Python 也不例外。

Python 中的字符串与其他编程语言中的字符串都担任着相同的角色，但 Python 和其他程序语言(C 语言或者 Java 语言)有所不同，它没有固定类型的字符串，和 PHP 中的字符串非常相似。在前面的章节中我们已经简单介绍了字符串，本章将介绍字符串的其他操作函数。

学习目标

- 掌握字符串的拼接。
- 掌握字符串格式化。
- 掌握截取字符串。
- 掌握比较字符串。
- 掌握搜索字符串。
- 掌握替换字符串。
- 掌握转换时间字符串。

6.1　邮箱注册系统

前面章节中学习的创建字符串知识，大家都知道这些功能满足不了我们的需求，接下来将会学习如何合理地应用和操作这些字符串。本节将介绍字符串的基础操作，包括字符串索引以及字符串转换等内容。

 视频教学：光盘/videos/06/基础操作.avi　　长度：17 分钟

6.1.1　基础知识——基础操作

在实际项目应用中，经常会用到对字符串进行简单的操作，比如使用 len()函数来获取字符串长度，通过使用+操作符对字符串进行合并，以及使用*操作符对字符串进行重复等功能。

```
#-*-coding:UTF-8 -*-
#用户注册
username=raw_input("请输入您注册的用户名：")
if len(username)>10:
    print "用户名不能大于 10 位";
else:
    print "注册成功！";
```

在上述代码中，获取用户输入的用户名，判断用户输入字符串是否在规定范围内，因此使用了 len()函数来获取字符串的长度。还可以通过使用+来对字符串进行拼接以达到想要的效果，详细代码如下：

```
username=raw_input("请输入您注册的用户名：")
message="恭喜您，注册成功！"
if len(username)>10:
    print "用户名不能大于 10 位";
else:
    print username+message;
```

当用户注册成功之后则给予友好的提示信息，在这里使用+将用户注册的名称和提示信息进行拼接，因此输出结果将会是用户名和提示信息拼接后的字符串。

还有就是使用*操作符将输出的字符串重复循环拼接指定的次数，然后将其显示出来。

```
stu=raw_input("请输入要打印的字符串：")
sex=5;
print stu*sex
```

上述代码输出结果将会使用户输入的字符串重复显示 5 次，同时拼接为一个字符串呈现出来。

6.1.2　基础知识——字符串索引和分片

字符串是字符的有序集合，能够通过其位置来获得具体的元素，因此可以通过位置来获取元素。Python 中字符串中的字符是通过索引提取的，索引从 0 开始，但不同于 C 语言的是可以取负值，表示从末尾提取，最后一个是-1，之前一个是-2，依此类推，即程序认为从结束处反向计数。

```
stu="hello word"
print stu[0]
print stu[1]
print stu[-1]
print stu[-2]
```

上述代码首先定义了字符串 stu，长度为 10 个字符。分别输出下标为 0、1、-1 和-2 的字符，运行结果如下：

```
h
e
d
r
```

其中 0 索引表示字符串的第一个字符，-1 索引则是字符串的最后一位字母。

分片的作用是可以用来从字符串中分离、提取一部分内容(子字符串)，还可以用于提取部分数据，分离出前缀、后缀等场合。当使用一对以冒号分隔的偏移索引字符串序列对象时，Python 就会返回一个新的对象，其中包含以这对偏移所标识的连续的内容。

左边的偏移被取作下边界(包含下边界在内)，而右边的偏移被认为是上边界(不包括上边界在内)。如果省略上下边界，则默认值分别对应 0 和分片对象的长度。

```
stu="hello word"
print stu[1:3]
print stu[1:]
print stu[:-1]
print stu[:]
```

在上述代码中，stu[1:3]获取的字符串是偏移为 1 和 2 的元素值。stu[1:]获取的结果是从第一个字符到上边界之间的所有字符，因为 stu[1:]中没有指定上边界值，因此默认情况下就是该字符串的长度。stu[:-1]则是获取最后一位字符之前的所有字符到下边界，因为下边界没有值，所以将会获取到字符串的第一个字符。而 stu[:]不用说大家都能猜出来，用来获取所有字符串的长度。该实例运行结果如下：

```
el
ello word
hello wor
hello word
```

学习到这里可能大多数读者对索引和分片仍不太了解，对它们之间的关系还有所迷惑，很难分清楚它们如何应用。下面我们将索引和分片进行简单的总结，希望能够帮助大家更快地理解和应用索引与分片。

1．索引获取特定偏移的元素

- 字符串中第一个元素的偏移为 0。
- 字符串最后一个字符的偏移为-1，倒数第二个则为-2，从右向左进行。
- stu[0]获取字符串中第一个字符。
- stu[-2]获取字符串中倒数第二个字符。

2．分片提取相应部分数据

- 通常上边界不包含在提取字符串内。
- 如果没有指定值，则分片的边界默认为 0 和序列的长度。
- stu[1:3]获取从偏移为 1 的字符一直到偏移为 3 的字符(不包括第三个偏移的字符)。
- stu[1:]获取从偏移为 1 的字符一直到字符串最后的字符(包括最后字符)。
- stu[:3]获取从偏移为 0 的字符一直到偏移为 3 的字符(不包括第三个偏移的字符)。
- stu[:-1]获取从偏移为 0 的字符一直到最后一个字符(不包括最后一个字符)。
- stu[:]获取从偏移为 0 直到末尾之间的所有元素。

可能大家在开发的过程中会遇到让自己百思不解的现象：在分片的中括号里面可能有 3 个值。请不要惊慌失措，这也是我们所学习的分片的一种。下面继续讲解分片的高级应用——第三个限制值。

在 Python 2.x 以上版本中，分片增加了一个可以选择的第三个限制值，它的作用是控制获取元素的步进。所谓步进就是每次获取列表的步长，比如我们需要通过分片来获取一个 1～9 字符中的偶数元素，那么使用分片的第三个限制值来控制显示是在好不过了，此时可以将步进设置为 2 即可，该语句的语法如下：

```
X[I:J:K]
```

在上述语法中，X 表示需要获取的元素字符串，I 表示获取偏移的值，J 表示最后获取的位置，而 K 则表示讲解的不进值，该值是可选值。

```
stu ="123456789"
print stu[1:-1:2]
```

在上述代码中，通过使用高级的分片来获取 1～9 之间的偶数字符，1 表示从偏移为 1 开始到最后一个元素(不包括最后一个元素)，2 是步进的长度，设置为 2 每次走两步，因此获取的值就是偶数值。

6.1.3 基础知识——字符串转换

众所周知，在大多数语言中，通常情况下字符串和数字是不可以进行相加的，同样 Python 也不允许字符串和数字直接相加。在 Python 中，+操作符具有双重作用：一是用来对字符串进行合并拼接，二是对数字进行累加计算，而对字符串和数字进行相加就会变得模棱两可了。

在有的情况下，必须让字符串和数字进行累加计算。例如，从后台获取一个订单统计报表，报表中保存了每个商品的单价，要求计算出订单中所有商品的价格总合。对单价是数字的很简单，但是有时候可能保存的是字符串和数字，怎么把字符串变为数字呢？这时需要用到本节所

讲的知识点：字符串转换函数。

```
num1="0"
mum2="0"
print "欢迎使用加法计算器";
num1=int(raw_input("请输入第一个计算值："));
num2=int(raw_input("请输入第二个计算值："));
print "计算结果为",num1+num2
```

在上述代码中，通过使用一个简单的加法计算器为例，首先通过使用 raw_input()函数来获取用户输入的计算值。在这里要注意的是，当获取到用户输入的值时该值的类型为 stu 类型的字符串，而大家都知道如果两个字符串之间使用+操作符只是将两次输入的值进行拼接，而不是想要的计算结果，因此通过使用 int()函数将用户两次输入的值转换为数字类型再进行计算，这样就达到想要的结果了。

但是上述实例中还有一定的问题，比如小数点如何计算，其实字符串转换不单单可以转换为数字类型，还可以转换到浮点型等。

```
print "欢迎使用加法计算器";
num1=float(raw_input("请输入第一个计算值："));
num2=float(raw_input("请输入第二个计算值："));
print "计算结果为",num1+num2
```

之后会深入学习内置的 eval 函数，它用于运行一个包含 Python 表达式的字符串。

```
print "欢迎使用加法计算器";
num1=raw_input("请输入第一个计算值：");
num2=raw_input("请输入第二个计算值：");
print "计算结果为",eval(num1+'+'+num2)
```

单个字符可以通过 ord 函数转换为对应的 ASCII 数值(整数)。chr 函数相反，可以将一个整数转换为对应的字符。

```
num1=int(raw_input("请输入 ASCII 值："));
print "元素结果为",chr(num1);
num2=raw_input("请输入元素");
print "ASCII 结果为",ord(num2);
```

6.1.4　实例描述

现在网络上有很多免费的邮箱系统，可是那些邮箱毕竟是别人的，大家有没有想过拥有一个属于自己的邮箱系统呢？本实例将通过使用字符串的基础操作来制作一个简单的邮箱注册系统。

6.1.5　实例应用

【例 6-1】使用字符串操作完成邮箱注册。

邮箱注册功能是邮箱系统中必不可少的部分，如果没有注册就无法进行添加用户，因此在

用户注册邮箱时需要用户输入邮箱账户、邮箱密码、用户名等信息，详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
print "【欢迎大家使用窗内网邮箱系统！】\n===========窗内 xxx@itzcn.com 邮箱注册
===========";
loginname=raw_input("请输入您要注册的用户名：");    #获取用户账号
loginpwd=raw_input("请输入您的登录密码");            #获取用户密码
name=raw_input("请输入您的真实姓名");                #获取用户姓名
yzm=int(raw_input("请输入验证码：3+3=？"));         #获取用户输入验证码
if yzm==6:                                          #判断用户输入验证码是否正确
    print "正在注册中，请稍后........"
    print "===========恭喜您注册成功了！===========\n",name+"：您好！","\n 您的邮
箱地址为",loginname+"@itzcn.com\n 您的密码为：",loginpwd+"\n 请保管好您的用户信息，
进行登录！";                                        #显示用户注册信息
else:
    print "验证码错误！"
    yzm=int(raw_input("请输入验证码：3+3=？"));     #验证码错误重新输入
```

在上述代码中，首先提示用户使用邮箱注册系统，再提示用户输入用户名、密码、真实姓名和验证码信息，并且使用 raw_input()函数来获取用户输入信息。大家知道，使用该函数获取的信息为 stu 字符串类型，因此在获取验证码时使用 int()将获取数据转换为数字类型。接着使用 if 语句判断用户输入的验证码是否正确，如果通过验证则提示用户注册成功，并且显示用户注册信息；如果判断失败则提示用户重新输入验证码。

6.1.6 运行结果

运行实例代码，结果如图 6-1 所示。



图 6-1 邮箱注册

6.1.7　实例分析


源码解析

在上述实例中，通过使用 raw_input()函数获取输入信息，同时将验证码信息进行了字符串转换为数字类型，然后进行判断，同时还使用+操作符将用户名和邮箱后缀进行拼接，充分应用了该节学习的对字符串的简单操作。

6.2　打印客户凭条

Python 格式化字符串作为计算机语言中常用的功能，在相关的实际操作应用中，它也有类似于 C 中的函数 printf()的格式输出标记。尽管这样可能用到非常复杂的表达式，但可以将一个值插入一个含有字符串格式符%s 的字符串中。


视频教学： 光盘/videos/06/字符串格式化.avi　　　　长度：5 分钟

6.2.1　基础知识——字符串格式化

Python 字符串格式化中的%操作符左边部分的“格式标记字符串”可以完全和 C 中的一致，学过 C 语言的读者肯定对这种语法不会陌生。右边的“值组”如果有两个以上的值则需要用小括号括起来，中间用逗号隔开。详细代码如下：

```
maildow="@itzcn.com"
loginname="duanshaozhi"
print "您的邮箱地址为：%s%s"%(loginname,maildow)
```

从上述代码可以得知，通过替换将%s 处的数据替换为 maildow 变量的值。如果需要插入的值不止一个，则需要在右侧用括号把它们括起来；如果插入的值只有一个，那么可以选择不用括号直接指定需要插入的变量或者值即可。

学到这里你可能会想：做了这么多工作只不过是为了做简单的字符串连接。没错，只不过字符串格式化不只是连接，甚至不仅仅是格式化，它也是强制类型转换。

插入的值类型可以为整型、浮点型、对象或者变量，但是如果目标左侧为%s 则是将它们转换为字符串存储。

目标左侧的%s 可用于设置格式化后的字符类型，以减少对转换后的元素类型再做转换。可以转换的字符串格式化代码如表 6-1 所示。

在形如%6.2f 的式子中，6 和 2 不能事先指定，它们会在程序运行过程中再产生。那么怎么输入呢？当然不能用%%d.%df 或%d.%d%f，但可以用%*.*f 的形式，即在%后面要输出的值组中包含两个*。比如'%*.*f% (6,2,2.345)相当于'%6.2f'%2.345。

这是本书到目前为止看到的最复杂的内容。如果记不住，或者不想那么耐烦，完全可以全

部用%s 代替，或者用多个+操作符来构造类似的输出字符串。这里的%真有点除法的味道，怪不得设计者会选择用%符号。

表 6-1　字符串格式化代码

代　码	意　义	代　码	意　义
%s	字符串	%e	浮点数字
%c	字符及其 ASCII 码	%E	浮点数字大写
%d	十进制整数	%f	浮点十进制数字
%u	无符号十进制整数	%g	浮点数字 e 或者 f
%o	八进制整数	%G	浮点数字 E 或者 F
%x	十六进制整数	%%	百分号标记
%X	十六进制整数大写	%n	用十六进制打印值的内存地址
%p	存储输出字符的数量放进参数列表的下一个变量中		

6.2.2　实例描述

凡是使用自动取款机取过钱的读者对这个词语已经非常耳熟。本实例将制作一个类似于银行自动取款机打印客户凭条的功能，该功能非常简单，只需用户输入卡号、密码、用户名和交易金额即可。

6.2.3　实例应用

【例 6-2】自动取款机客户凭条打印。

首先获取用户输入的卡号和密码进行登录，登录成功后，提示用户进行取款业务。首先需要输入用户的真实姓名，然后输入需要交易的金额，当交易成功后，将会把你所交易的信息打印出来，详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
print "欢迎使用 XX 银行自动取款机！";
car=raw_input("请输入您的卡号：");
pwds=raw_input("请输入您的密码：");
print "登录成功！进行取款业务。";
name=raw_input("请输入您的用户名：");
money=int(raw_input("请输入交易金额："));
print "正在交易中，请稍后......\n 交易成功，请查收您的客户凭条！";
print "【XX 银行自动取款机客户凭条】\n============================";
print "%s 您好！\n 您的卡号%s\n 本次交易金额为%f\n 欢迎下次使用！"%(name,car,money);
print "============================";
```

在代码中，通过使用 raw_input()来获取用户执行交易时输入的用户信息，最主要的则是本节讲的格式化字符串，将用户的信息插入到打印信息中。可使用%s 将字符串插入信息中，使

用%f 将信息格式为浮点类型显示出来，但是该替换的值必须为整型数字。

6.2.4　运行结果

运行程序，然后输入用户信息，结果如图 6-2 所示。

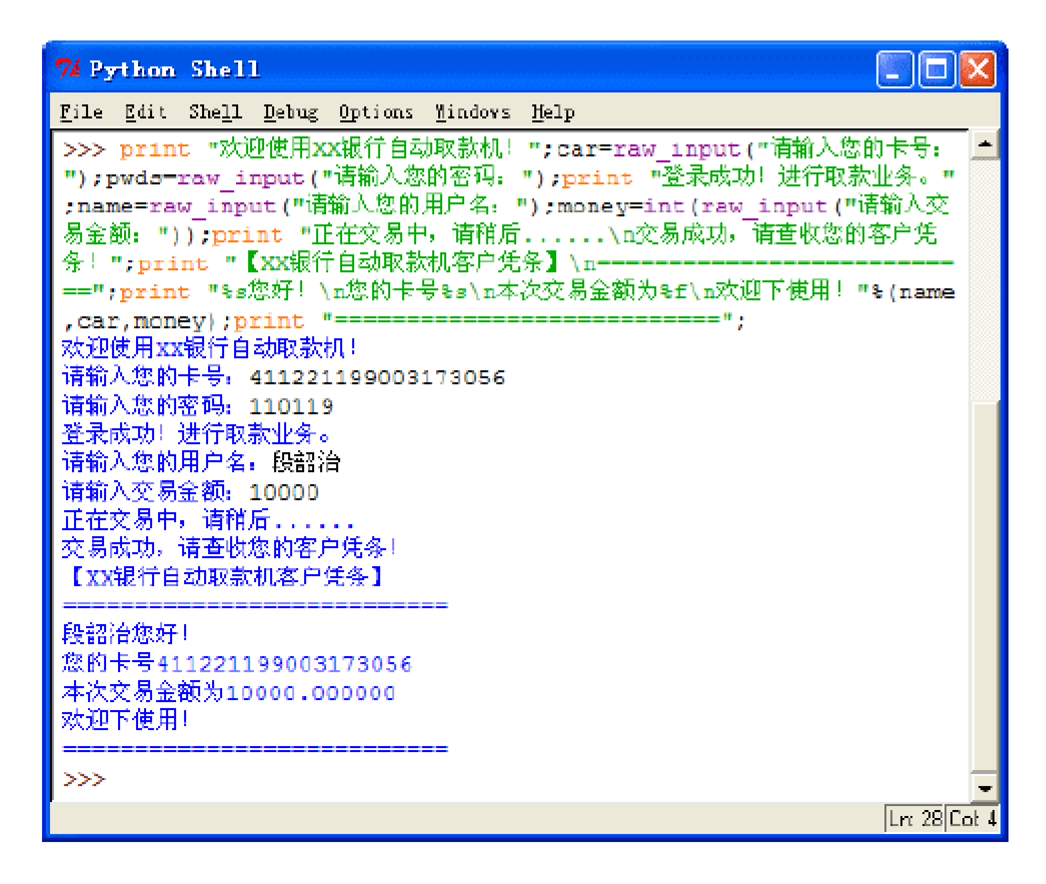


图 6-2　客户凭条打印

6.2.5　实例分析


源码解析

在上述实例中，通过使用%s 和%f 完成了客户凭条的打印功能。使用%f 格式化的结果只是浮点类型的元素，但是传入的值必须为整型的数字。同样，还可以使用其他的字符串格式化代码对需要的程序作出相应的字符串处理。

6.3　列车路线查询系统

合并字符串的函数有多个，学习到这里大家已经知道可以使用+操作符对字符串进行合并，但是仅仅使用+操作符是远远不够的。本节将介绍另一种字符串合并的函数 join()。


视频教学：光盘/videos/06/ join()函数.avi　　　　长度：6 分钟

6.3.1　基础知识——join()函数

join()函数也是操作字符串的一个非常重要的函数，它和我们下一节将要讲到的 split()函数互为逆函数。为什么说互为逆函数呢？因为 split()函数可用来通过某个标识符将字符串拆分为集合，而 join()函数则是把一个集合中所有的值按照自定义的分隔符连接起来，详细代码如下：

```
letters1=['a','b','c','d','e']
letters2='f','g','h','i','j'
sep="-"
print sep.join(letters1);
print "/".join(letters2);
```

在上述代码中，声明了一个带有中括号的集合和一个普通的集合，其中第一种则是使用声明标识符最后使用标识符变量调用 join()函数合并字符串，第二种则是直接使用标识符调用 join()函数，这两种不同的方法都可以达到想要的效果，运行结果如下：

```
a-b-c-d-e
f/g/h/i/j
```

6.3.2　实例描述

下面通过一个简单的列车路线查询系统为案例，使用 join()方法来完成该功能。首先在列表中保存每条路线的站点，当用户输入目的城市的拼音简写时，程序将相应的路线通过 join()方法截取并且返回在显示控制台提供用户参考。

6.3.3　实例应用

【例 6-3】列车路线查询系统。

在该实例中，首先声明了 3 条由郑州到达某城市的路线，分别使用集合将到达的每个站点保存为一条记录，然后声明了拼接的字符串格式，接受用户输入信息，并调用 join()方法进行拼接。

```
smx=['郑州','上街','巩义','偃师','洛阳','新安县','渑池','三门峡 '];
bj=['郑州','新乡','新乡','石家庄','保定','北京西'];
ly=['郑州 ','巩义','偃师','洛阳'];
sy="=>>";
print "欢迎使用由郑州开往各地列车路线查询系统";
area=raw_input("请输入到底地点地名简称例：北京 bj\n");
print "您到达目的地需要经过的车站如下：\n",sy.join(smx)
```

在上述代码中，声明集合时还可以将中括号去掉，不妨碍程序的运行，最后则调用拼接字符串的 join()方法传入参数，该参数则是集合体。

6.3.4　运行结果

运行代码，结果如图 6-3 所示。

```
Python Shell
File  Edit  Shell  Debug  Options  Windows  Help
    ****************************************************************

IDLE 1.2.4
>>> smx=['郑州','上街','巩义','偃师','洛阳','新安县','渑池','三门峡 '];bj=[
'郑州','新乡','新乡','石家庄','保定','北京西'];ly=['郑州 ','巩义','偃师','洛
阳'];sy="=>>";print "欢迎使用由郑州开往各地列车路线查询系统";area=raw_input(
"请输入到底地点地名简称例：北京bj\n");print "您到达目的地需要经过的车站如下：\n"
,sy.join(smx)
欢迎使用由郑州开往各地列车路线查询系统
请输入到底地点地名简称例：北京bj
smx
您到达目的地需要经过的车站如下：
郑州=>>上街=>>巩义=>>偃师=>>洛阳=>>新安县=>>渑池=>>三门峡
>>>
Ln: 18 Col: 4
```

图 6-3　列车路线系统

6.3.5　实例分析

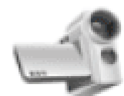


源码解析

从上述实例可以看到，需要添加的集合元素必须是字符串。但是如果在使用过程中需要遍历某些特殊字符，则可以通过使用转义符将特殊字符进行转义，否则程序将会以操作符进行处理。

6.4　获取邮箱用户名

字符串截取在实际案例开发中也是必不可少的功能之一，有时需要通过某些特殊字符串将某字符串截取为一个集合并且显示在列表中。本节将介绍两种不同的字符串截取函数 split()和 strip()。



视频教学：光盘/videos/06/ split()和 strip()函数.avi　　长度：9 分钟

6.4.1　基础知识——split()函数

该函数是字符串处理中非常重要的一个函数，用来将字符串通过某些标识符进行分割为序列，该函数和我们上节讲到的 join()函数相反，该函数用来分割而上节讲解到的则是用来拼接。在本节的讲解过程中，大家可以上节内容的逆思路进行学习。

```
#-*-coding:UTF-8 -*-
#Python 模板
strs1="A=>>B=>>C=>>D=>>E=>>F";
strs2="A/B/C/D/E/F";
strs3="A B C D E F";
print "分割=>>标识符字符串",strs1.split("=>>");
print "分割/标识符字符串",strs2.split("/");
print "无标识符",strs3.split();
```

如果分隔符是一个或者多个空格，那么你就不必给出任何参数，直接使用 str.split()即可。需要注意的是，这种分隔方法对中间为空的项会忽略，也就是说如果你要对多行文本逐行 split()，假如你期望每行都被分隔为 10 列，但是如果某一列为空，那么你可能只会得到 9 列。

6.4.2 基础知识——strip()函数

Python 中的 strip 用于去除字符串的首位字符。同理，lstrip 用于去除左边的字符，rstrip 用于去除右边的字符。这 3 个函数都可传入一个参数，指定要去除的首尾字符。



传入的是一个字符数组，编译器去除两端所有相应的字符，直到没有匹配的字符为止。

```
strs = 'saaaay yes no yaaaass'
print strs.strip('say')
```

通过上述代码可以得知，strs 将会依次被去除首尾在['s'，'a'，'y']数组内的字符，直到字符不在数组内为止，因此将会输出 yes no 字符串。

同理，前面提到的 rstrip()和 lstrip()函数的工作原理相同，但要注意的是，如果函数中没有传入参数，则会去除字符串中首尾的空格。

```
#-*-coding:UTF-8 -*-
#Python 模板
strs = 'duanshaozhi@126.com'
print strs.strip('duanshaozhi')
print strs.lstrip('duanshaozhi')
print strs.rstrip('@126.com')
```

上述代码中使用了 strip()函数，输出结果是@126.com，而使用 lstrip()函数输出的结果则是去除左边的字符@126.com，使用 rstrip()函数将会去除右边的字符 duanshaozhi。

6.4.3 实例描述

该实例通过用户输入邮箱来获取邮箱登录用户名，使用简单的字符串截取功能，首先获取用户输入的邮箱字符串，判断用户输入的邮箱类型，然后根据用户输入的不同邮箱类型来判断截取邮箱的登录名称，从而达到用户快速登录的结果。

6.4.4 实例应用

【例 6-4】截取邮箱用户名。

该实例首先通过使用 raw_input()函数来获取用户输入邮箱地址，通过截取字符串来获取邮箱地址的用户账号，然后提示用户输入用户密码，完成一键登录功能。

```
#-*-coding:UTF-8 -*-
#Python 模板
print "欢迎使用邮箱快速登录系统";
mail=raw_input("请输入您的 126 邮箱地址：");
```

```
username=mail.strip("@126.com");
password=raw_input("请输入您的登录密码：");
print username,"您好，欢迎登录 126 邮箱！";
```

代码中，可以看到使用 strip()函数来截取用户输入邮箱地址，该函数传入参数设置为登录邮箱类型，在这里设置为“@126.com”在使用该函数截取时将会提取参数字符串中之前的字符串然后进行操作。

6.4.5　运行结果

运行 Python 程序，放入执行代码，结果如图 6-4 所示。



图 6-4　邮箱快速登录

6.4.6　实例分析



源码解析

实例非常简单。使用一个简单的字符串截取函数 strip()来获取邮箱用户登录名，但是在实际应用中要注意使用的位置，如果使用不当，可能给程序带来不必要的麻烦。

6.5　上传图片格式判断

比较函数在 Python 中也是比较常用的函数之一，学过 Java 语言的读者可能都知道：在 Java 中，如果需要比较两个字符串就需要使用 equals()函数，而在 Python 中可以直接使用运算符来进行判断，比较两个字符串是否相同，但要注意的是，在比较两个字符串时，如果类型不同，那么比较的结果肯定不同。

如果要想比较一个字符串中的某一部分内容，则可以先通过截取字符串再进行比较；如果要比较字符串开头和结尾部分，那么建议大家选择使用 startswith()和 endswith()函数。下面将对这两个函数进行详细讲解。



视频教学：光盘/videos/06/ startswith()和 endswith()函数.avi　　长度：6 分钟

6.5.1 基础知识——startswith()函数

该函数用来判断一个文本是否以一个字符串开始，如果判断成立则返回 True，否则返回 False，该函数语法如下：

```
startswith(substring[,start[,end]])
```

上述代码中 startswith()函数有 3 个参数，其中 substring 为与原字符串开头部分比较的字符串，start 参数是开始比较的位置，而 end 则是结束的位置。

```
#-*-coding:UTF-8 -*-
#Python 模板
strs = 'itzcn'
strs.startswith("it");
strs.startswith("z",2);
strs.startswith("zcn",2,5);
```

6.5.2 基础知识——endswith()函数

该函数的使用方法和 startswith()函数类似，该函数用来判断一个文本是否以一个字符串结束，如果判断成立则返回 True，否则返回 Fasle。

```
endswith(substring[,start[,end]])
```

函数 endswith()的参数和返回类型类似于 startswith()，不同的是，endswith()函数是从原字符串结尾开始查找判断。

```
#-*-coding:UTF-8 -*-
#Python 模板
strs = 'itzcn'
strs.endswith("cn");
strs.endswith("n",4);
strs.endswith("cn",3,5);
```

搜索字符的位置同样可以使用 len()函数来获取当前字符串的长度来定位。

6.5.3 实例描述

在大多数宣传性网站后台都会有上传图片功能，上传图片也是很多网站必不可少的一项功能。实现该功能的好处是减少了用户直接登录服务器上传图片的麻烦，用户可以直接通过网站后台进行上传图片。本实例将对图片的格式进行截取、判断，防止用户将非法图片格式上传至服务器。

6.5.4 实例应用

【例 6-5】图片上传功能。

该实例通过使用比较字符串函数来实现一个简单的图片上传功能，首先获取用户输入图片

的路径并截取图片后缀，然后使用字符串比较函数来判断当前图片的格式和系统指定允许上传格式是否一致，如果通过验证，则执行上传功能。

```
#-*-coding:UTF-8 -*-
#Python 模板
    print "========欢迎使用图片上传系统========"
filename=raw_input("请输入需要上传的图片路径地址");
if filename.endswith(".gif") or filename.endswith(".jpg"):
    print "%s 图片格式正确，正在上传中....."%filename
else:
    print "图片格式不正确，请上传 GIF 或者 JPG 格式图片";
```

上述代码使用了 raw_input()来获取上传图片的地址，然后使用 if 判断及 endswith()方法截取最后字符串格式与指定字符串进行比较，如果返回 True 则提示用户图片格式正确进入上传阶段，否则提示图片格式错误。

6.5.5　运行结果

运行代码，输入上传图片地址，结果如图 6-5 所示。

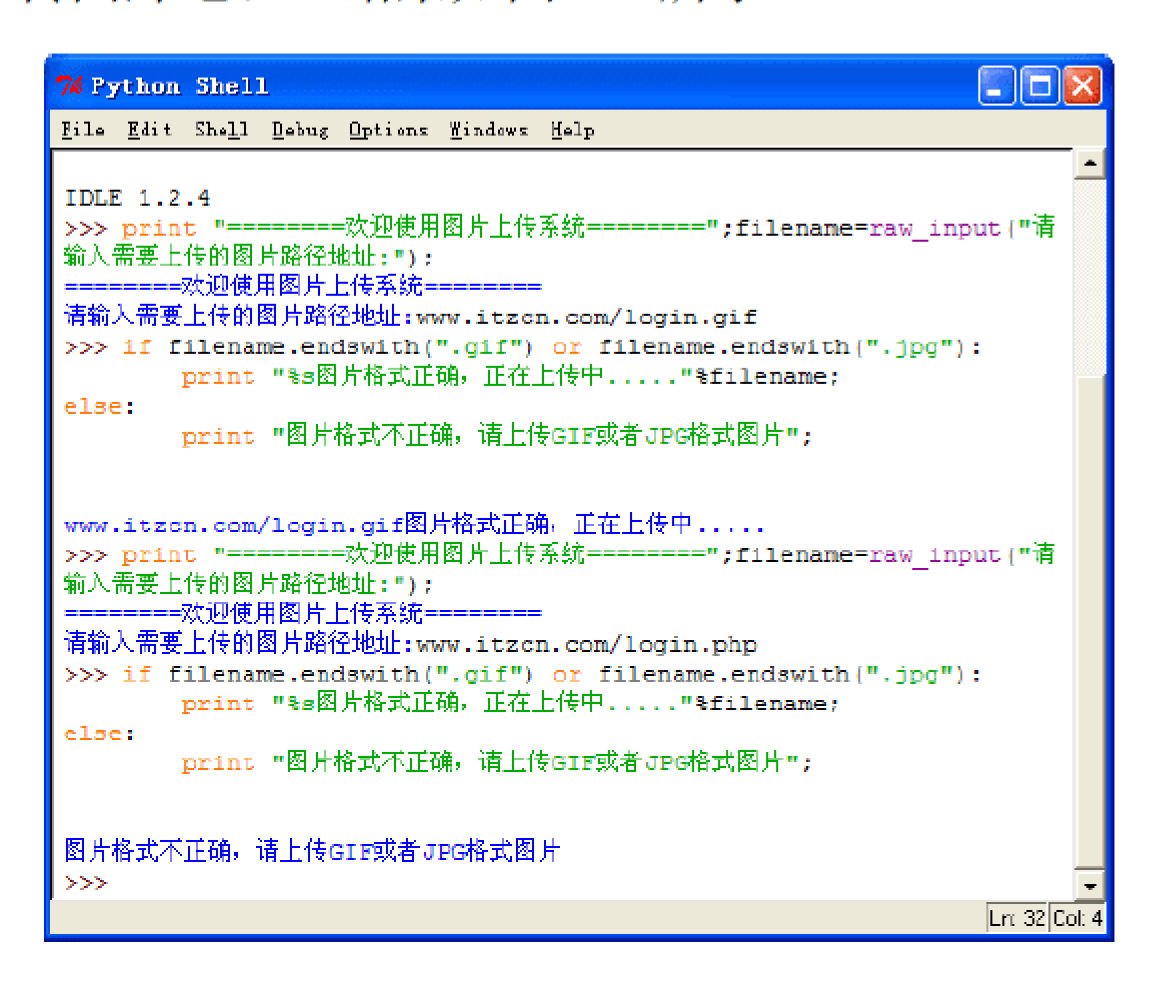


图 6-5　图片上传

6.5.6　实例分析

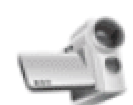


源码解析

在上述实例中，通过使用 endswith()函数来判断字符串的结尾部分和指定字符串是否相同，从而达到判断文件格式的功能，相反可以用 startswith()函数来对一段文字的前部分和指定字符串进行比较，经常用到文件通过名称排序等功能上。

6.6 邮箱用户名长度验证

字符串搜索也是一个很重要的功能。在实际应用中，字符串搜索功能就成了家常便饭。话虽如此，但是在使用时要适可而止。Python 定制了一套关于字符串搜索的函数，其中包含 find()和 rfind()函数。下面将对该函数做详细讲解。

 视频教学：光盘/videos/06/ find()函数.avi　　长度：7 分钟

6.6.1 基础知识——find()函数

字符串查询函数在其他编程语言中非常普遍，例如 Java 中的 indexOf()就是用来查询字符串中指定字符第一次出现的下标位置，注意下标由 0 开始。该函数相当本节所讲解的 Python 中的字符串查询函数 find()，它们有着相同的工作原理。同样，Java 中的 lastIndexOf()函数用来查询字符串相同字符的位置，但是它是从右往左进行查询。在 Python 中也有类似于 Java 中 lastIndexOf()的函数，rfind()则是 Python 中的从右往左进行查询的函数，其工作原理也与之类似。

find()和 rfind()函数的语法类似，其定义如下：

```
find(substring [,start [,end]])
```

其中 find()函数有 3 个参数值，第一个参数 substring 表示等待查询的字符串或字符。第二个参数 start 表示待查询字符串的开始坐标，而第三个参数表示待查询字符串的结束坐标，返回结果则是一个数字类型，如果返回-1，则表示查找字符串不存在。

```
substring="Welcome to the itzcn";
print "从左边开始的第一个 t 坐标为：",substring.find("t");
print "从右边开始的第一个 t 坐标为：",substring.rfind("t");
```

以上代码通过使用 find()和 rfind()函数来查询字符串中 t 出现的坐标位置，其中从左边开始查询输出的结果为 8，从右边开始查询的结果为 16。



待查询字符串的第一个字符坐标为 0，同时字符串中空格也表示一个字符，占有一个字符位置。

6.6.2 实例描述

网站注册是每个商业性网站必不可少的功能之一，注册的主要作用就是保存用户信息，以便和客户交流、沟通。注册信息中可能会有邮箱地址，邮箱地址也属于网站注册必不可少的信息。为了防止不法邮箱被录入，判断邮箱格式是否正确非常重要。

本实例通过一个简单的用户注册来验证用户输入邮箱地址长度是否合法。

6.6.3　实例应用

【例 6-6】邮箱地址长度验证。

实例通过输入用户基本信息和邮箱，获取邮箱地址，并判断邮箱的用户名长度是否合法，然后提示用户。

```
#-*-coding:UTF-8 -*-
#Python 模板
print "=========Welcome to the itzcn=========\n 窗内网>>会员注册";
username=raw_input("请输入用户登录: ");
userpwd1=raw_input("请输入密码: ");
userpwd2=raw_input("请输入确认密码: ");
mail=raw_input("请输入 126 邮箱地址: ");
print "正在验证中,请稍等......";
if mail.rfind("@")>18 or mail.rfind("@")<6:
    print "您输入邮箱地址不合法,请重新输入邮箱地址";
    mail=raw_input("请输入 126 邮箱地址: ");
else:
    print "验证通过,正在注册中.....";
```

在此代码中，同样使用了 raw_input()函数来获取用户输入的邮箱地址，然后调用 rfind()函数来查询字符串中@的位置，这样就可以获取用户名的长度，如果长度大于 18 或者小于 6，则提示用户重新输入正确的邮箱地址。

6.6.4　运行结果

运行代码，用户输入信息，输入正确和错误邮箱地址时的运行结果如图 6-6 所示。



图 6-6　邮箱地址验证

6.6.5 实例分析



源码解析

通过实例，大家可能对 find()和 rfind()函数的应用有了比较深刻的印象。在实际案例开发中，使用该函数来验证一些数据信息也是常见的。但是在应用中，可能查询的对象字符串是中文，此时应记住一个汉字占两个字符。

6.7 文章内容过滤

字符串替换是指 Python 操作字符串时经常会碰到的问题，在 Python 中要想实现字符串替换有两种方法：一是使用正则表达式，二是使用本节主要讲解的字符串自带的函数。本节将介绍字符串替换函数 replace()和 translate()，正则替换字符串函数将在以后的讲解中介绍。



视频教学：光盘/videos/06/ replace()和 translate()函数.avi　　**长度**：6 分钟

6.7.1 基础知识——replace()函数

replace()函数是 Python 中的替换字符串函数之一，该函数可以指定替换的次数。以下是 replace()函数语法：

```
replace(substring,newsubstring[,max])
```

在该语法中，replace()函数有 3 个参数，其中参数 substring 表示被替换的字符串对象，newsubstring 则是应替换的内容，max 表示替换的次数，默认为替换所有字符。

```
#-*-coding:UTF-8 -*-
#Python 模板
substring="Welcome to the connection window";
print "替换前：",substring;
print "替换后：",substring.replace("connection window","itzcn")
print "更改替换次数",substring.replace("c","C",2);
```

以上代码将字符串中所有和 connection window 相同的内容全部替换为 itzcn，以及将小写 c 替换为 C，同时只替换两个相同的字符。

6.7.2 基础知识——translate()函数

translate()函数和上一节讲解的 replace()函数一样，都是字符串替换函数，它们的功能相同，但也有一定的区别。translate()函数只处理单个字符，同时可以进行多个字符的替换，而且替换的效率比 replace()快。

translate 函数可以根据 deletechars 删除字符，可以根据 table 转换字符，但这个 table 必须给出 256 个字符的转换规则。先不说这 256 个字符是哪些，仅仅输入这个 table 就够麻烦了。在此可使用 string 模块提供的 maketrans 函数，这里就不做实例了。

6.7.3　实例描述

大多数读者可能都了解网站备案，其原因就是怕你的网站里面有非法内容，但是大多数站长最头疼的就是诬陷，非法内容其实并不是站内管理人员发表的，可能是某些用户发表的，为此站长可背上了黑锅。有没有拯救的措施呢？肯定有。其一就是禁止用户发表评论，可是有些网站还必须允许用户发表评论。该怎么办？过滤器将对用户输入的内容进行过滤，它将用到 Python 中的 translate()替换函数，用于对用户输入的不法内容或者站内禁止的字符串进行替换。

6.7.4　实例应用

【例 6-7】文章过滤器。

该实例以窗内网留言系统做示范，首先获取用户输入的留言标题和留言内容，然后通过使用字符串替换函数将敏感字符和禁止字符替换为星号，然后显示输出。详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
print "=========欢迎使用窗内网留言系统=========\n 窗内网>>留言板";
title=raw_input("请输入留言标题：");
content=raw_input("请输入留言内容：");
newtitle=title.replace("过滤器","***");
newtitle=newtitle.replace("后台管理密码","***");
newcontent=content.replace("过滤器","***");
newcontent=newcontent.replace("坏蛋","***");
print "您的留言内容为：";
print "标题：",newtitle;
print "内容：\n%s"%newcontent;
```

在此实例代码中，首先使用 raw_input()方法将用户输入的留言标题和留言内容进行保存，然后调用 replace()函数，将字符串中存在指定字符串的内容全部替换为星号。

6.7.5　运行结果

结果如图 6-7 所示。

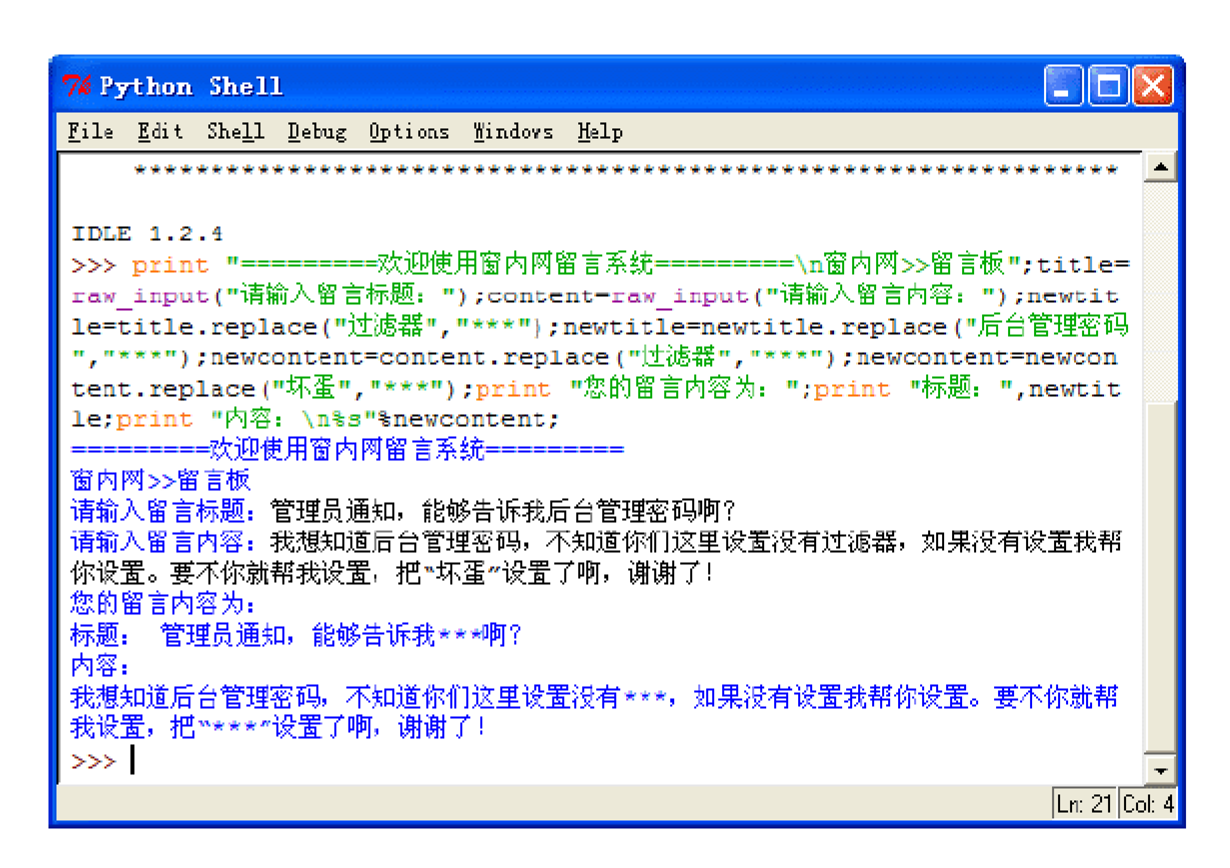


图 6-7　留言板

6.7.6　实例分析



源码解析

实例使用了 replace()函数来实现字符串替换，从而完成了留言过滤作用，这样大大减少了不法信息的侵入。该函数不管是字符串还是单个字符都可以进行替换，如果仅对一个字符做替换，那么建议大家使用 translate()函数。

6.8　转换时间字符串strptime()函数

将一个字符串转换为时间类型在其他编程语言中也有相应的方法，Python 也有转换的方法，唯一不同的是我们需要通过两次转换才能完成想要的结果。



视频教学：光盘/videos/06/ strptime()函数.avi　　　　长度：6 分钟

将字符串转换为时间类型需要通过两次转换，将使用到 time 模块和 datetime 类。下面是转换的整个过程。

首先需要调用 strptime()函数，将字符串转换为一个元组，该函数的语法格式如下：

```
strptime(substring,format)
```

其中，strptime()函数有两个参数，第一个参数为需要转换的字符串，第二个参数则是输出的时间格式。该函数的返回结果是一个保存时间的元组。

接下来是第二次转换，将表示日期的变量传递给 datetime()函数进行转换，该函数位于 datetime 类下，其基本语法如下：

```
datetime(year,month,day)
```

在上述语法中，有 3 个参数，其中第一个 year 代表年，第二个参数 month 表示月，第三个参数 day 则表示日，这 3 个参数是必不可少的。同时还可以有时、分、秒参数可选。该函数

返回结果是一个 datetime 类型的变量，时间格式与字符串格式转化时定义的格式一致，如表 6-2 所示。

表 6-2　时间格式化命令

命　令	描　述	命　令	描　述
%a	星期几的简写，如星期三为 Web	%e	在两字符域中，十进制表示的每月的第几天
%A	星期几的全称，如星期三为 Wednesday	%F	年-月-日
%b	月份的简写，如 4 月份为 Apr	%g	年份的后两位数字，使用基于周的年
%B	月份的全称，如 4 月份为 April	%G	年份，使用基于周的年
%c	标准日期的时间串	%h	简写的月份名
%C	年份的后两位数字	%H	24 小时制的小时
%d	十进制表示的每月的第几天	%I	12 小时制的小时
%D	月/天/年	%r	12 小时的时间
%R	显示小时和分钟：hh:mm	%S	十进制的秒数
%T	显示时分秒：hh:mm:ss	%u	每周的第几天，星期一为第一天
%U	第几年的第几周，把星期日作为第一天	%x	标准的日期串
%X	标准的时间串	%%	百分号

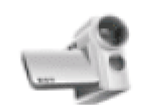
下面是一个简单地将字符串转换为时间的示例，详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
import time;
import datetime;
print time.strftime("%Y 年%m 月%d 日 %X",time.localtime());
t= time.strptime("2011-3-8","%Y-%m-%d");
y,m,d=t[0:3]
print datetime.datetime(y,m,d);
```

在上述代码中，首先使用 inport 关键字引入了 time 模块和 datetime 类，接下来使用 time 模块中的 strftime()函数来获取当前系统时间，然后进行了简单的格式转换。time 模块下还有一个函数 strftime()，用来对字符串进行格式化。该函数有两个参数，第一个是需要格式化的字符串，第二个是时间的格式，此时返回的结果为一个元组。最后调用 datetime 类中的 datetime()函数对字符串进行格式显示。

6.9　会员注册系统

正则表达式用于字符串处理、表单验证等场合，实用高效。初学 Python，对 Python 的文字处理能力有很深的印象，除了 str 对象自带的一些方法外，就是正则表达式这个强大的模块了。对初学者来说，要用好这个功能还真有点难度，我花了好长时间才摸出了一点门道。Python 语言的 re 模块对基本的正则表达式做了许多有益的改进。对需要处理文本的程序员来说，必须对正则表达式有一个全面、深入的认识。

 视频教学：光盘/videos/06/正则表达式.avi　　长度：11 分钟

6.9.1 基础知识——正则表达式简介

正则表达式是一种可以用于模式匹配和替换的强大工具。在几乎所有的基于 UNIX/Linux 系统的软件工具中都能找到正则表达式的痕迹，例如 Python 或 PHP 脚本语言。此外，JavaScript 这种客户端脚本语言也提供了对正则表达式的支持。现在，正则表达式已经成为一个通用的概念和工具，被各类技术人员所广泛使用。

正则表达式由字母、数字和特殊字符组成，别看里面有特殊字符，构成这些特殊的正则表达式的主要因素还是它们。

特殊字符在正则表达式中也是比较重要的。在正则表达式中，特殊字符分为元字符和定位符等。其中，元字符是正则表达式中一类特殊意义的字符，它的主要作用就是用来描述其前导字符在被匹配的对象中出现的方式，表 6-3 仅列举了一些特殊字符的含义。

表 6-3　正则表达式中的特殊字符

字　符	描　述
^	表示匹配的字符必须在最前边
$	与^类似，匹配最末的字符
*	匹配*前面的字符 0 次或 n 次
+	匹配+号前面的字符 1 次或 n 次
?	匹配?前面的字符 0 次或 1 次
.	(小数点)匹配除换行符外的所有字符
(x)	匹配 x 并记录匹配的值
x|y	匹配 x 或者 y
{n}	这里的 n 是一个正整数。匹配前面的 n 个字符
{n,}	这里的 n 是一个正整数。匹配至少 n 个前面的字符
{n,m}	这里的 n 和 m 都是正整数。匹配至少 n 个最多 m 个前面的字符
[xyz]	字符列表，匹配列表中的任一字符。可以通过连字符-指出字符范围
[b]	匹配一个空格
b	匹配一个单词的分界线，比如一个空格
B	匹配一个单词的非分界线

上述表格只是正则表达式的一些特殊字符的定义，例如[]字符则是用来匹配一个范围，该字符经常会用到一些邮箱或者电话、身份证号的验证上。



在上述表格中^与[^m]中的^所表示的意义是不相同的。

在使用正则表达式限制一些字符的范围时少不了定界符，可以利用{}来控制字符重复的次数。例如，我们需要编写一个用来控制邮箱的正则表达式，详细代码如下：

```
/^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/
```

在上述代码中，用来控制邮箱中@前半部分可以使用任意字母或者任意来代替，而后面则只由字母和数字组成。

6.9.2　基础知识——使用正则表达式

在 Python 中，re 模块集成了正则表达式的全部功能，提供一系列方法用于正则表达式的匹配和替换，这些函数使用一个模式字符串作为它们的第一个参数。表 6-4 列出了 re 模块中的常用函数。

表 6-4　re模块常用函数

函　数	描　述
re.match()	re.match 尝试从字符串的开始匹配一个模式
re.search()	re.search 函数会在字符串内查找模式匹配，直到找到第一个匹配然后返回，如果字符串没有匹配，则返回 None
re.sub()	re.sub 用于替换字符串中的匹配项
re.split()	可以使用 re.split 来分割字符串
re.compile()	可以把正则表达式编译成一个正则表达式对象
re.findall()	re.findall 可以获取字符串中所有匹配的字符串
escape(pattern)	匹配字符串中的特殊字符



在 re 模块中，大多数函数都会有 flags 参数，flags 参数的作用就是用来设置匹配的附加选项。

比如需要忽略字符串中的大小写或者是否支持多行匹配等附加匹配选项，都可以通过设置该参数来完成。表 6-5 列出了该参数的一些匹配规则。

表 6-5　re模块匹配规则

名　称	描　述
IGNORECASE	忽略文中的大小写
LOCALE	处理字符集本地化
MULTILINE	是否支持多行匹配
DOTALL	匹配一些特殊标记，例如使用.匹配\n 等字符
VERBOSE	忽略正则表达式中的空格或者换行等字符
UNICODE	使用 Unicode 编码

下面将对 re 模块中的重用函数进行讲解。

1. re.match()函数

以下代码是该函数的基本语法。

```
match(string[, pos[, endpos]]) | re.match(pattern, string[, flags])
```

该函数将从 string 的 pos 下标处开始尝试匹配。如果 pattern 匹配结束时仍可匹配，那么将会返回一个 Match 对象；如果匹配过程中 pattern 无法匹配，或者匹配未结束就已到达 endpos，则返回 None。

pos 和 endpos 的默认值分别为 0 和 len(string)，其中 re.match()无法指定这两个参数。参数 flags 用于编译 pattern 时指定匹配模式。



该函数并不是完全匹配。如果 pattern 结束时 string 还有剩余字符，仍然视为成功。想要完全匹配，可以在表达式末尾加上边界匹配符$。

2. re.search()函数

该函数用于查找字符串中可以匹配成功的子串，其语法如下：

```
search(string[, pos[, endpos]]) | re.search(pattern, string[, flags])
```

该函数从 string 的 pos 下标处开始尝试匹配 pattern，如果 pattern 匹配结束时仍可匹配，则返回一个 Match 对象；若无法匹配，则将 pos 加 1 后重新尝试匹配，直到 pos=endpos 时仍无法匹配则返回 None。

pos 和 endpos 的默认值分别为 0 和 len(string)，其中 re.match()无法指定这两个参数。参数 flags 用于编译 pattern 时指定匹配模式。

3. re.sub()函数

该函数的作用是用来替换正则表达式中所匹配的选项内容，找到 re 匹配的所有子串，并将其用一个不同的字符串替换，该函数的语法如下：

```
sub(repl, string[, count]) | re.sub(pattern, repl, string[, count])
```

该函数被替换后，如果匹配模式没有发现，则 string 将被原样返回。可选参数 count 是模式匹配后替换的最大次数，它必须是非负整数，缺省值 0 表示替换所有的匹配。

4. re.split()函数

将字符串在 re 匹配的地方分片并生成一个列表，该函数的语法如下：

```
split(string[, maxsplit]) | re.split(pattern, string[, maxsplit])
```

该函数在 re 匹配的地方将字符串分片，并返回这些部分的列表。它类似于字符串的 split()方法，但是它提供更多的定界符。split()只支持通过空格和固定字符来分片。正如你期望的，也有一个模块级函数 re.split()。

5. re.compile()函数

该函数也接受一个可选的标识参数，用来使各种特别的特性和语法变种起作用，该函数的语法如下：

```
re.compile(strPattern[, flag])
```

这个函数是 Pattern 类的工厂方法，用于将字符串形式的正则表达式编译为 Pattern 对象。第二个参数 flag 是匹配模式，其取值可以使用按位或运算符|表示同时生效。

6. re.findall()函数

找到所有正则表达式匹配的子字符串，并把它们作为一个列表返回，该函数的语法如下：

```
findall(string[, pos[, endpos]]) | re.findall(pattern, string[, flags])
```

下面通过一个简单的实例看一下如何使用该函数。

```
#-*-coding:UTF-8 -*-
#Python 模板
import re;
p=re.compile(r'\d+');
print p.findall('one1tow2three3four4');
```

在代码中首先引入 re 模块库，将正则定义在模块中，最后调用 findall()函数进行字符匹配，运行结果如下：

```
['1', '2', '3', '4']
```

7. re.escape(pattern)函数

re 模块还提供了一个函数 escape(string)，用于在 string 中的正则表达式元字符(如*/+/?等)之前加上转义符再返回，它在需要大量匹配元字符时有那么一点用处。

6.9.3　实例描述

本实例将以用户注册系统为基础，制作一个窗内网注册系统。该系统将通过使用正则表达式来对系统中用户输入的信息进行验证，如果用户输入信息不合法，则提示用户重新输入；如果验证成功，则显示用户注册的基本信息。

6.9.4　实例应用

【例 6-8】会员注册系统。

该系统首先定义变量，用来接受用户输入的基本信息，接下来通过使用 Python 中的正则表达式的方法对用户输入的邮箱格式进行严格的验证，其中邮箱里必须包含有@和.字符，并且字符长度也有一定的限制，详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
import re;
print "==================欢迎使用窗内网会员注册系统==================";
name = raw_input("请输入用户名：\n");
pwds1= raw_input("请输入登录密码：\n");
pwds2= raw_input("请确认密码：\n");
text = raw_input("请输入邮箱地址：\n");
m = re.match(r"^([a-z0-9A-Z]+[-|\\.]?)+[a-z0-9A-Z]@([a-z0-9A-Z]+
(-[a-z0-9A-Z]+)?\.)+[a-zA-Z]{2,}$", text);
if m:
    print name,"您好！您的账号已经注册成功！\n 您的邮箱地址为：",text+"\n 请保管好您的
账号信息！";
```

```
else:
    print '邮箱格式不正确，请重新输入邮箱地址！';
    text = raw_input("请输入邮箱地址：\n");
    m =
re.match(r"^([a-z0-9A-Z]+[-|\\.]?)+[a-z0-9A-Z]@([a-z0-9A-Z]+(-[a-z0-9A-Z]+)
?\.)+[a-zA-Z]{2,}$", text);
```

在实例代码中，首先需要引入 re 模块，通过使用 import 关键字进行导入，导入后则使用 raw_input()函数来获取用户输入的基本信息。接受用户输入的邮箱信息，然后进行正则表达式验证，调用 re 模块中的 match()函数进行验证。该函数有两个参数，第一个参数是需要验证的正则表达式规则，第二个参数是需要验证的字符串。通过验证和判断，如果返回值为 None 则表示验证失败，否则验证通过给出提示信息。

6.9.5　运行结果

运行上述实例，结果如图 6-8 所示。

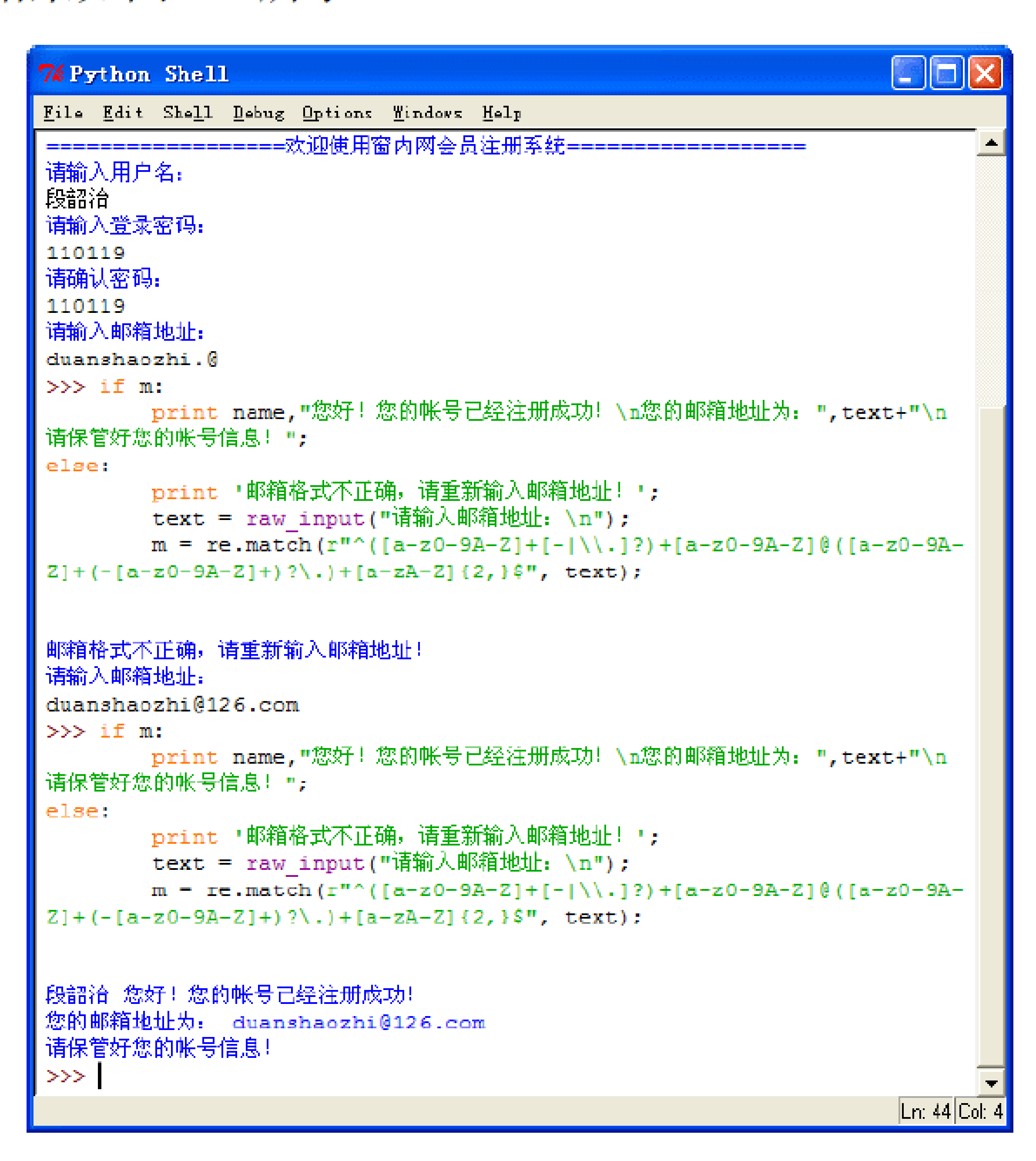


图 6-8　会员注册系统

6.9.6　实例分析



源码解析

实例中主要使用了 re.match()函数来对字符串进行规则验证，并返回相应的结果。re 模块中还有其他常用的函数，例如 re.split()函数用来替换字符串中的数据，re.sub()函数则用来截取字符串，这些函数都是在实际项目中经常用到的。

6.10　常见问题解答

6.10.1　格式化字符串%号问题



格式化字符串%号问题？

网络课堂：http://bbs.itzcn.com/thread-15227-1-1.html

今天一大早起来编写了一段 Python 代码，其中用到了格式化字符串。通过使用%将数据显示在控制台上，可是怎么运行都会出现错误，希望大侠帮忙解决一下，感激不尽。

```
#-*-coding:UTF-8 -*-
#Python 模板
name="段韶治";
sex="男";
age="100";
mail="duanshaozhi@126.com";
print "【%s 用户基本信息】"%name;
print "姓名：%s\n 性别：%s\n 年龄：%s\n 邮箱：%s"%name,sex,age,mail;
```

错误信息如图 6-9 所示。

```
Python Shell
File Edit Shell Debug Options Windows Help
    ****************************************************************

IDLE 1.2.4
>>> name="段韶治";sex="男";age="100";mail="duanshaozhi@126.com";print
 "【%s用户基本信息】"%name;print "姓名：%s\n性别：%s\n年龄：%s\n邮箱：%s"
%name,sex,age,mail;
【段韶治用户基本信息】

Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    name="段韶治";sex="男";age="100";mail="duanshaozhi@126.com";print
 "【%s用户基本信息】"%name;print "姓名：%s\n性别：%s\n年龄：%s\n邮箱：%s"
%name,sex,age,mail;
TypeError: not enough arguments for format string
>>> |
Ln: 19 Col: 4
```

图 6-9　错误信息

有点疑惑的是，在代码中第一次使用 print 函数时能够正常显示，而在使用第二次时就出

现了错误信息。

【解决办法】通过我对你的代码的观察，你的基础知识不怎么好哦。之所以第一个 print 函数中的数据能够正常显示而第二个则报错，主要原因在于第二个 print 函数中有多个%参数值，在这种情况下，字符串后面的%中应该使用小括号将你传入的参数括起来。在书中已将讲解过，如果有多个参数就使用小括号，你肯定没有好好阅读。正确的代码如下：

```
print "【%s 用户基本信息】"%name;
print "姓名：%s\n 性别：%s\n 年龄：%s\n 邮箱：%s"%(name,sex,age,mail);
```

6.10.2　无法对字符串进行拆分



使用正则表达式问题？

网络课堂：http://bbs.itzcn.com/thread-15228-1-1.html

在编写一段代码时遇到一个很严重的错误，我居然在使用正则表达式对字符串进行拆分时出现错误，希望大家帮我看一下那个错误在哪里，下面是我做的一个非常简单的例子的代码。

```
#-*-coding:UTF-8 -*-
#Python 模板
text = "Welcome+to+the+connection+window"
print re.split(r'+',text)
```

报错信息如图 6-10 所示。

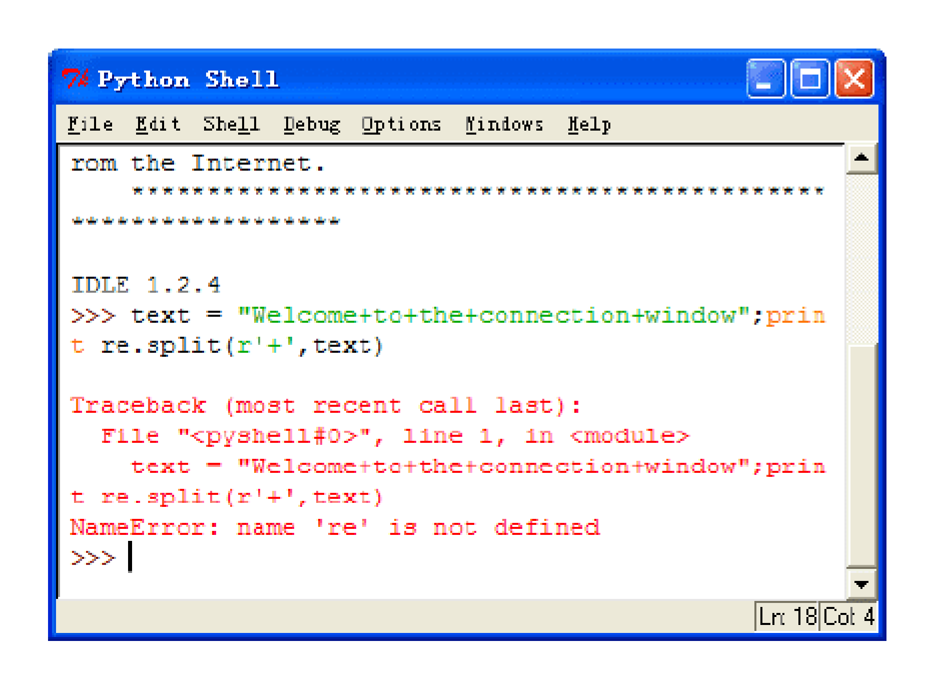


图 6-10　错误提示

【解决办法】在帮你解决问题之前先帮你纠正一个错误，在你的代码中不是一个错误，我却看到了两个错误，导致代码无法运行。首先是第一个错误，在使用 re 模块时，需要将 re 模块导入程序中，而在你的代码中却没有。第二个错误是，在使用正则表达式对字符串进行拆分时，你使用了 re 模块下的 split()函数，但是你需要拆分的关键字正好是正则表达式中的操作符，因此出现了冲突。你可以通过\操作符对该字符进行转义，正确代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
import re
text = "Welcome+to+the+connection+window"
print re.split(r'\+',text)
```

6.11　习　　题

一、填空题

(1) 插入的值类型可以是整型、浮点型、对象或者变量，那么目标左侧应该为__________。

(2) 通常使用 Python 内置的__________函数来对一个集合中的数据进行合并。

(3) 如果需要判断一个字符串是否以某个字符结束，那么通常使用 Python 中的__________函数。

(4) 在 Python 中使用__________函数来获取当前系统时间。

(5) 在使用 strptime()函数进行时间格式化时__________命令用来显示时、分、秒。

二、选择题

(1) 在将字符串进行格式化为 ASCII 码时，我们使用__________代码进行格式化。

A. %s　　B. %d　　C. %c　　D. %o

(2) find()函数用来查询字符串中是否包含某个字符，下面选项中__________选项是没有匹配字符所返回的值。

A. 0　　B. −1　　C. 1　　D. Flase

(3) 在以下代码中，运行结果正确的选项为__________。

```
"Welcome to read this book".replace("o","O",2);
```

A. `'WelcOme tO read this bOOk'`　　B. `'WelcOme to read this bOok'`

C. `'WelcOme tO read this book'`　　D. `'Welcome to read this book'`

三、上机练习

上机练习： 电话号码验证。

该实例通过接受用户输入信息来判断用户电话号码是否合法，如果用户输入的电话号码长度不正确或者没有-便视为非法电话号码，将提示用户重新输入电话号码。验证完毕，将电话号码的区号和号码进行拆分，显示到控制台上。实例要求通过使用 Python 中的正则表达式来对电话号码进行验证和拆分，运行结果如图 6-11 所示。

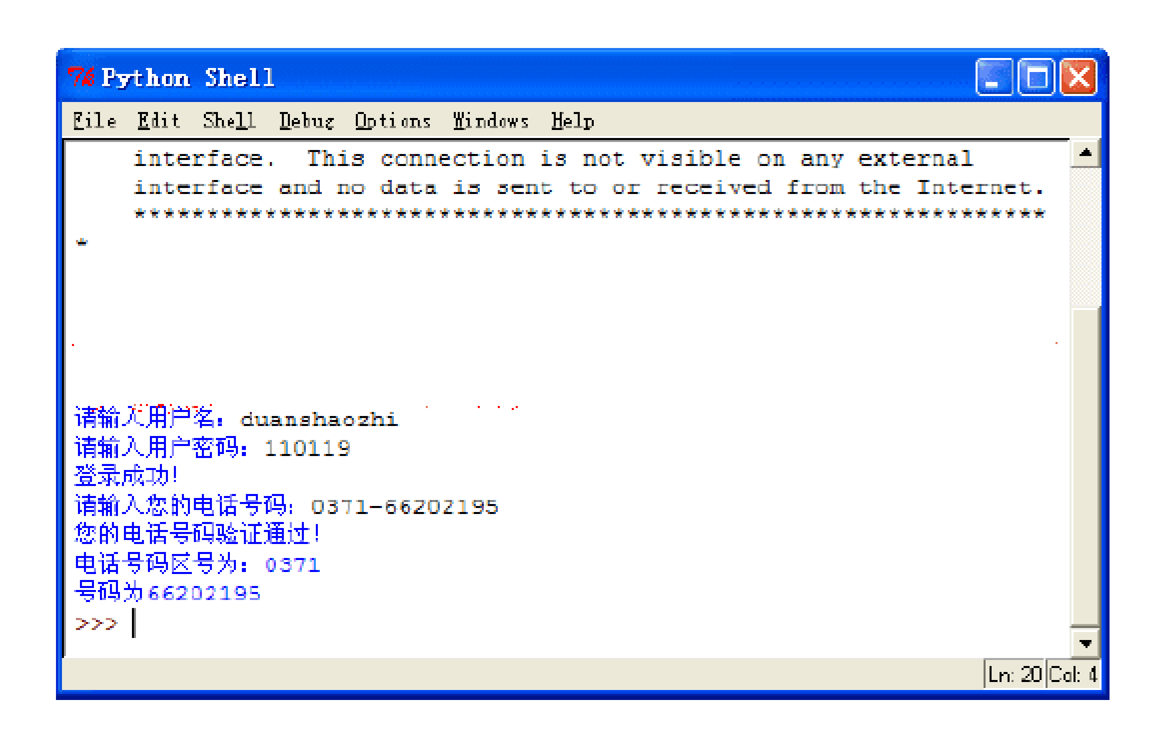


图 6-11　电话号码验证



第 7 章　面向对象编程

内容摘要

面向对象编程提供了一种新的思维方式，使得软件设计的焦点不再是程序的逻辑流程，而是软件或者程序中对象以及对象之间的关系。使用面向对象思想进行程序设计，能够更好地设计软件架构，维护软件模块，易于框架和组件的重用。

本章首先针对面向对象编程的三大要素：封装、继承和多态作详细讲解，接着详细介绍Python 类的属性和方法以及其他特性，最后简单介绍在新式类中引入的属性和方法。

学习目标

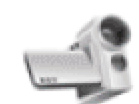
- 熟练掌握封装、多态的使用。
- 掌握如何创建类和对象。
- 熟练使用类的方法和属性。
- 熟练掌握继承的使用。
- 了解类的其他特性。
- 掌握新式类中属性和方法的使用。

7.1　面向对象编程

面向对象编程中的对象(object)可以看作是对现实世界实体的模拟，由现实实体的过程或者信息来定义。一个对象可以被认为是一个把数据(属性)和程序(方法)封装在一起的实体，也可以认为是这个程序产生该对象的动作或它接受到外界信号的反应。面向对象的优点主要包括以下 3 个方面。

- 多态(Polymorphism)：说明可以对不同类的对象使用相同的操作。
- 封装(Encapsulation)：对外部世界隐藏对象的工作细节。
- 继承(Inheritance)：以普通的类为基础建立专门的类对象。

在许多面向对象程序设计的介绍中，由于封装和继承被用作现实世界中对象的模型，因此一般会先介绍封装和继承。当然，这种思想很好，但是在我们这些初学者看来，很容易就能理解封装和继承，对多态却始终处于一种迷迷糊糊的状态，对面向对象理解得不够深刻。在本章中，首先介绍多态，以便激发你对面向对象程序设计的兴趣。下面我们就来看一下多态。

 **视频教学**：光盘/videos/07/面向对象的编程.avi　　　　长度：14 分钟

7.1.1　基础知识——多态

多态按字面的意思就是多种状态。多态意味着即使不知道变量所引用的对象类型是什么，仍能对它进行操作，而该变量会根据对象类型的不同表现出不同的行为。例如，我们随便打开一个购物网站，该网站创建了一个在线支付系统，那么程序会从系统的其他部分获得商品的价格，然后使用信用卡在线支付即可。

当程序获得商品时，我们首先想到如何具体的体现它们。例如，需要将它们作为元组接收，代码如下：

```
('name',12.50)
```

如果程序只需要体现商品的名称和价格，而且该商品可以打折，也就是说商品在卖出之前价格会逐渐降低，于是我想将商品先放入购物车中，等到价格最便宜时，再单击“支付”按钮进行购买。这样，使用元组就不能满足我们的需要。

想要实现这个功能，就使代码每次访问价格的时候，对象必须检查当前的价格，因此价格不能固定在元组中。某些聪明的程序员可能想到使用十六进制的字符串来表示价格，然后存储在字典中的某个键下面，访问价格的时候，只需要更新函数即可。但是如果有些人希望为该键下面的价格增加新的字典类型呢？你仍然可以再次更新定义好的函数，只是这种工作需要多长时间？每次有人要实现价格对象的不同功能时，该怎么办？显然这是不灵活而且不切实际实现多种行为的代码编写方式。

到底该怎么办？我们可以让对象自己进行操作。听起来很模糊，但仔细想一下，这样做会轻松很多。每个新的对象都可以检索和计算自己的价格并且返回结果，这样我们只需向它询问价格即可。这就要靠多态来实现了。

1. 多态和方法

程序得到一个对象，但并不知道它怎么实现，它很可能有多种形状。你要做的就是询问价格，这就可以了。实现方法如下：

```
object.getPrice()
```

绑定到对象特性上面的函数称为方法，我们见过的有字符串、列表和字典方法。请看下面一段代码。

```
>>> 'duanchunyang'.count('g')
1
>>> 'duanchunyang'.count('u')
2
>>> 'duanchunyang'.count('n')
3
>>> [1,2,2,4,'dcy'].count(4)
1
```

对变量来说，不需要知道它是字符串还是列表，就可以调用它的 count 方法，根本不用管它是什么类型，只需要提供一个字符作为参数即可。

2. 多态的多种形式

当不知道对象是什么类型，但需要对象做点什么的时候，都会用到多态，这不仅仅限于方法，还有很多内建运算符和函数都能体现多态的性质，例如：

```
>>> 25+25
50
>>> 'duan'+'chun'+'yang'
'duanchunyang'
>>>
```

从上述代码运行的结果可以看出，这里的加号运算符对数字和字符串都能起作用。上述代码中两个对象相加可以使用下面的代码代替。

```
def myadd(x,y):
       return x+y
print myadd(1,2)
print myadd('duanchun','yang')
```

在上述代码中，其中的参数可以是任何支持加法的对象。如果我想编写打印对象长度的方法，那么只需要对象具有长度即可。下面就是我测试对象长度的代码。

```
def lengthObject(myobject):
       print "该对象是使用 repr 方法：",repr(myobject),'该对象的长度是：
',len(myobject)
lengthObject('duanchunyang')
lengthObject(['d','u','a','n','c','h','u','n','y','a','n','g'])
```

运行程序，执行结果如下：

```
>>>
该对象是使用 repr 方法： 'duanchunyang' 该对象的长度是： 12
该对象是使用 repr 方法： ['d', 'u', 'a', 'n', 'c', 'h', 'u', 'n', 'y', 'a', 'n',
'g'] 该对象的长度是： 12
>>>
```

从上述结果可以看出，函数 repr 是多态特性的代表之一，对任何对象都可以使用。

7.1.2 基础知识——封装

所谓封装，即将抽象得到的数据和行为(或功能)相结合，形成一个有机的整体，也就是将数据与操作数据的源代码进行有机的结合，形成类，其中数据和函数都是类的成员。其目的是隐藏对象的属性和实现细节，对外公开接口，在程序中控制属性的读和修改的访问级别。

封装的概念听起来是不是有点像多态呢？需要注意的是，封装并不等于多态，多态可以让用户对不知道是什么类的对象进行方法调用，封装则不用关心对象是如何构建的而直接进行使用。接下来，我们使用多态而不用封装写个例子，代码如下：

```
class MyFZ:
    name='mxl'
    def setName(self,name):
        self.name=name
    def getName(self):
        return self.name
```

在上述代码中，我们在类 MyFZ 中定义了一个全局变量 name，另外还定义了两个方法 setName 和 getName。

创建 MyFZ 类的实对象，代码如下：

```
>>> myfz=MyFZ()
>>> myfz.setName('duanchunyang')
>>> myfz.getName()
'duanchunyang'
>>>
```

如上述代码所示，声明一个实例对象 myfz，然后使用 setName 方法为对象设置一个值，最后调用 getName 方法，打印输出，这是最完美不过了。但是如果我们将变量存储到全局变量 name 中，是不是意味着在使用 MyFZ 类实例的时候，全局变量 name 的值会有所改变。下面就是存储到全局变量中的代码。

```
>>> myfz=MyFZ()
>>> myfz.name='you are the one'
>>> myfz.getName()
'you are the one'
>>>
```

从上面程序得出的结果可以看出，我们不得不关心全局变量 name 的内容，实际上需要确保程序对它不会有任何更改。

怎么办呢？别急，我们可以使用封装，将名称封装到对象内，然后将其作为特性存储。这样在调用方法时就不用关心其他东西了。例如，它是否干扰了全局变量。

封装和其他方法一样，特性是对象内部的变量，如果不用全局变量而是用特性重写类，那么它会像下面这样工作。代码如下。

```
>>> myfz=MyFZ()
>>> myfz.setName('ILOVEYOU')
>>> myfz.getName()
'ILOVEYOU'
>>>
```

到目前为止，我们还不能确定值是否还存储在全局变量中，那么再创建另一个对象，代码如下：

```
>>> myfzg=MyFZ()
>>> myfzg.setName('WELCOME')
>>> myfzg.getName()
'WELCOME'
>>>
```

可以看到新对象的名称已经正确设置，这才是我们期望的。接下来看一下第一个对象怎么样了，代码如下：

```
>>> myfz.getName()
'ILOVEYOU'
>>>
```

瞧，名字还在，这是因为对象有自己的状态。对象的状态由它的特性(名称)来描述，其中对象的方法可以改变它的特性。就像将一大堆方法捆绑在一起，并且给予它们访问变量的权利，它们才可以在方法调用之间保持所保存的值。

7.1.3　基础知识——继承

继承是一个懒惰的行为，当程序员不想将同一段代码写很多次时，我们可以使用函数来避免这种情况。但是如果已经有一个类，另外你还想建立一个和这个类非常类似的类，在该类中有可能添加几个方法或者几个属性，你又不想将原始类的代码全部复制过去，在这种情况下使用继承无疑是最好的选择。

这部分内容将在后面的章节中详细介绍。

7.2　创建自定义类

当我们在使用 C#语言开发网站或者系统时，首先需要对网站或者系统中的各个模块进行分析，接着建立数据库表，之后根据数据库表来定义实体类，注意在这里所提到的实体类描述的就是一个对象。另外，我们还可以将所有的对象放到一个类中，并且在类中对不同的对象进行描述，这就是本节将要介绍的内容。


视频教学： 光盘/videos/07/类和对象.avi　　　　长度：9 分钟

7.2.1 基础知识——类和对象

创建类(class)实际上就是对某种类型的对象定义变量和方法的原型。类表示的是对现实生活中一类具有共同特征事物的抽象，是面向对象编程的基础。

类是对某个对象的定义。它包含有关对象动作方式的信息，包括它的名称、方法、属性和事件。由于它不存在于内存中，因此它本身并不是对象。当程序运行需要引用类的代码时，就会在内存中创建一个类的新实例，即对象。虽然只有一个类，但能以这个类在内存中创建多个相同类型的对象。

有一句话这样说：万物皆对象。所谓对象可以认为是一件事、一个实体、一个名词，可以想象有自己标识的任何东西。你也可以这么说：所有的对象都属于某一个类，即类的实例。

例如：在空旷的田地里，你可以看到很多树，那么这些树就可以认为是一个实例。你看到的可能是大叶黄扬，也可能是梧桐树，而这里的大叶黄杨、梧桐树就是一个对象，并且该对象有自己的特征。关于树类和树对象的关系，下面的流程图将会给我们最好的说明，如图 7-1 所示。

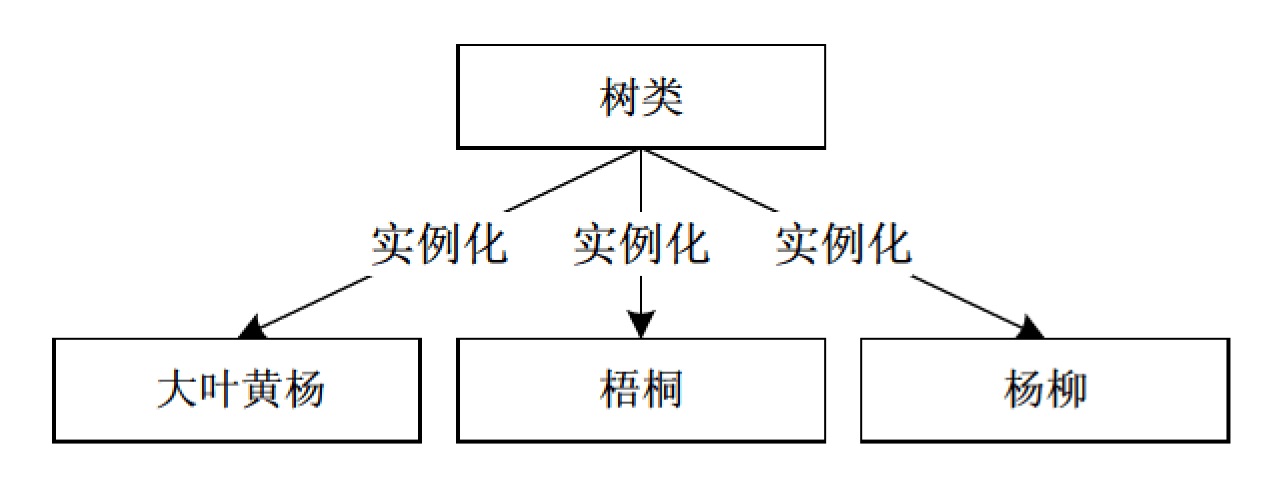


图 7-1　类和对象的关系

通过上面的介绍，你是否对类和对象有了进一步的理解呢？我想是肯定的。下面我们总结一下 Python 语言中类和对象的区别，以便在后面能够有目的地学习。

在 Python 语言中，对象包括特性和方法。特性是作为对象的一部分变量，方法是存储在对象内的函数。方法总是将对象作为自己的第一个参数，这个参数被称为 self，而类则代表对象的集合。在这里每个对象都有一个类，目的是定义它的实例会用到的方法。

下面来看一下在 Python 语言中如何创建类。

1. 类的创建

在其他编程语言中，我们使用 class 关键字来创建类，那么在 Python 语言中是不是同样用 class 关键字呢？答案是肯定的。下面我们就来看一下在 Python 中如何定义类。

首先来看一下定义类的语法格式。

```
class 类名:
    def 方法名 1 (参数名):
        pass
```

从上述语法来看，类必须使用 class 关键字来定义，接着是类名，其中 pass 占位符替代的是由一系列的属性和方法组成的主体。

下面我们通过一个例子来说明，代码如下：

```
class Person:
    def getName (self):
        print 'my name is:duanchunyang'
    def getAge(self):
        print 'My age is:20'
    def getHoppy (self):
        print 'My hoppy is:lvyou'
```

在上述代码中，我们定义了一个名称为 Person 的类，由于每个人都有不同名称、年龄以及爱好等，因此在 Person 类中定义了 3 个方法，分别是 getName()、getAge()和 getHoppy()，并且在类的方法中至少有一个参数 self，否则程序运行时会出现错误。



对于学过 Java 的人来说，很容易看出：在 Python 中 self 相当于 this，只不过 Python 中的 self 需要明确写出，才能使属性的读取更加明显。

2．对象的创建

前面提到对象是类的实例，那么对象的创建过程也可以说是类实例化的过程。在 Java 语言中，实例化一个对象需要使用 new 关键字，而在 Python 语言中则不需要使用 new 保留字。下面创建一些实例，然后调用该实例中的方法。

```
>>> person=Person()
>>> person.getName()
my name is:duanchunyang
>>> person.getAge()
My age is:20
>>> person.getHoppy()
My hoppy is:lvyou
>>>
```

从上述结果来看，将 Person 类实例化成一个名称为 person 的实例，之后使用 person.的方式调用该类中的方法，显然在调用 getName()方法时，没有传入参数，而且程序运行正常。这是否可以说明 self 的作用了？在调用 person 的 getName、getAge 和 getHoppy 方法时，person 自动将自己作为一个参数传入方法中，因此我们称它为 self。



类的方法必须有一个 self 参数，但是方法被调用时，不用传递这个参数。有关 self 参数的知识，在后面的章节中会详细介绍。

7.2.2　实例描述

你喜欢看故事书吗？如果喜欢，那么一定不会对“故事大杂烩”这个词感到陌生。你喜欢为故事中的人物写同人吗？如果喜欢，你一定不会因为我将“故事大杂烩”改名为“世界大杂烩”而丧失兴趣。在“故事大杂烩”中，包含了各种类型的故事，那么在“世界大杂烩”中同样有不同类型的对象，其对象的特征也会在细节中体现出来。下面就来看一下自定义的“世界大杂烩”类。

7.2.3 实例应用

【例 7-1】创建自定义类。

(1) 创建一个名称为 MyClass.py 的文件。

(2) 在文件 MyClass.py 中添加代码。

```
mystr=raw_input('请输入你想要知道的对象')
class MyWorld:
    #定义的人对象
    def person (self):
        self.mytalk='我可以用语言来表达'
        self.mylimbs='也可以用肢体语言来表达'
        self.myeyes='你可以眉目传情吗'
        print '我是人，因此我可以
%s,%s,%s'%(self.mytalk,self.mylimbs,self.myeyes)
    #定义的猪对象
    def pig (self):
        self.mytalk='哼哼哼哼'
        self.myspecialty='吃饭，睡觉'
        self.mymaster='谁对我好，谁就是我的主人'
        print '我是猪，我的特点就是
%s,%s,%s'%(self.mytalk,self.myspecialty,self.mymaster)
    #定义的公鸡对象
    def rooster (self):
        self.mywork='在天蒙蒙亮的时候打鸣'
        self.mymotto='所谓闻鸡起舞，说的就是我'
        print '我是公鸡，我可以%s,%s'%(self.mywork,self.mymotto)

if __name__=='__main__':
    myworld=MyWorld()
    if mystr=='人':
        myworld.person()
    elif mystr=='猪':
        myworld.pig()
    elif mystr=='公鸡':
        myworld.rooster()
    else:
        print '不好意思，该类中没有录入该对象'
```

(3) 保存修改好的代码。

7.2.4 运行结果

执行程序，当输入你需要了解的对象为“人”时，执行结果如下：

```
>>>
请输入你想要知道的对象人
我是人，因此我可以我可以用语言来表达,也可以用肢体语言来表达,你可以眉目传情吗
>>>
```

当你输入需要了解的对象为“猪”时，执行结果如下：

```
>>>
请输入你想要知道的对象猪
我是猪，我的特点就是哼哼哼哼,吃饭，睡觉,谁对我好，谁就是我的主人
>>>
```

当你输入你需要了解的对象为“公鸡”时，显示效果如下：

```
>>>
请输入你想要知道的对象公鸡
我是公鸡，我可以在天蒙蒙亮的时候打鸣,所谓闻鸡起舞，说的就是我
>>>
```

7.2.5　实例分析



源码解析

在本实例中，定义了一个名称为 MyWorld 的类，在类中定义了 3 个对象，分别是人、猪和公鸡。针对这 3 个对象的特点，分别使用 person、pig 和 rooster 方法来描述。接着实例化对象，然后根据输入的内容判断要调用的方法，最后将方法中的内容打印输出。

7.3　模拟水果成熟的过程

在使用 Java 语言开发网站或者系统时，类以及类中的属性和方法都是必不可少的，否则类与类之间方法和属性的调用就无从谈起。所谓面向对象编程语言，即是将所有的对象都归属的一个类中，而这些对象可以有自己的属性和方法，这和现实中的万物皆对象是完全相符的。本节就来看一下对类中属性和方法的调用。

**视频教学**：光盘/videos/07/属性和方法.avi　　　　**长度**：14 分钟

7.3.1　基础知识——属性和方法

众所周知，类是由属性和方法组成的。其中属性是对数据的封装，而方法则是对象具有的行为。在 Java 语言中，对属性和方法的公有和私有都是通过访问修饰符来区分的，例如公有属性和私有属性分别使用访问修饰符 public 和 private。那么在 Python 语言中有没有访问修饰符呢？当然没有，Python 中的构造函数、析构函数、私有属性(方法)、公有属性(方法)都是通过名称的约定来辨别的。

1．属性

接触过 Java 的人肯定知道，类的属性分为私有属性和公有属性。如果只需要让内部的函

数访问其属性，那么声明属性时，只需加上访问修饰符 private 即可。相应的，如果想让任何函数都可以访问该属性，则使用 public 访问修饰符便可解决。在 Python 语言中，并没有公有和私有权限的修饰符，仅仅取决于属性的名称。如果函数、方法或者属性的名称以两个下划线开始，则说明为私有类型。相反，如果没有以两个下划线开始，则表示为公有属性。



在 Java 中，还有一个受保护类型修饰符 protect。在 Python 语言中不存在这种类型。

在 Python 语言中，属性分为静态属性和实例属性。

何为静态属性？在 Java 中，被 static 修饰的属性被称为静态变量，该变量可以直接被类调用。在 Python 中，这样的静态变量被称为静态属性，也可以认为是类属性。

1) 公有属性

实例属性即以 self 作为前缀的属性，如果在类的方法中定义的变量没有使用 self 作为前缀声明，那么该变量就是一个普通的局部变量。

下面我们来看一个例子，代码如下：

```
class Fly:
    price=123        #公有的类属性
    def __init__ (self):
        self.direction='开往北京的火车'    #公有的实例属性
        zIng='候车厅中，人很多'             #局部变量
if __name__=="__main__":
    print Fly.price
    fly=Fly()              #实例化 Fly 类
    print fly.direction
    Fly.price=fly.price+10
    print '候车厅怎么样'
    print '火车票的价格上涨后的结果是：'+str(fly.price)
    myfly=Fly()
    print '我的理想价格是：'+str(myfly.price-20)
```

在上述代码中，定义了公有的类属性 price，并设置其初始价格为 123。接着在__init__方法中定义了一个名称为 direction 的实例属性，并且赋值，之后声明了一个普通的局部变量。

保存并运行程序，执行结果如下：

```
123
开往北京的火车
候车厅怎么样
火车票的价格上涨后的结果是：133
我的理想价格是：113
```

如果使用实例化对象引用局部变量 zIng，则执行结果如下：

```
123
开往北京的火车

Traceback (most recent call last):
  File "F:\Python\第 7 章\person.py", line 11, in <module>
    print '候车厅怎么样'+fly.zIng
AttributeError: Fly instance has no attribute 'zIng'
```

从结果中出现的错误来看，该局部变量不能被 Fly 类的实例化对象所使用。



在 Java 中，静态变量只能被类调用，而在 Python 中类和对象都可以访问类属性。

接下来看一下私有属性在 Python 中的应用。

2) 私有属性

将公有的实例属性 direction 修改成私有__direction，代码如下：

```
class Fly:
    price=123       #公有的类属性
    def __init__ (self):
        self.__direction='开往北京的火车'      #私有的实例属性
        zIng='候车厅中，人很多'               #局部变量
if __name__=="__main__":
    print Fly.price
    fly=Fly()        #实例化 Fly 类
    print fly.__direction
```

保存代码，执行结果如下：

```
123

Traceback (most recent call last):
  File "F:\Python\第 7 章\person.py", line 9, in <module>
    print fly.__direction
AttributeError: Fly instance has no attribute '__direction'
```

从上述结果可以看出，私有的属性不能被实例化对象访问。那么该如何访问方法中的私有属性呢？别急，Python 提供了直接访问私有属性的方式，其语法格式如下：

```
实例化对象名._类名__私有属性名
```

接下来，我们修改一下代码，如下所示：

```
class Fly:
    price=123       #公有的类属性
    def __init__ (self):
        self.__direction='开往北京的火车'      #私有的实例属性
        zIng='候车厅中，人很多'               #局部变量
if  name  ==" main  ":
    print Fly.price
    fly=Fly()        #实例化 Fly 类
    print fly._Fly__direction
```

其运行的结果如下：

```
>>>
123
开往北京的火车
>>>
```

从上述结果可以看出，使用 fly._Fly__direction 方式可以直接访问私有属性，这种方式常用于开发阶段的测试和调试，而获得实例属性值的一般做法是定义相关的 get 方法(将在下一节中介绍)。

3) 数据属性

上面介绍了如何使用实例化对象调用类属性以及方法中的私有属性，另外还有一种属性也可以被实例对象调用，称为数据属性。它不需要预先定义，当数据属性初次被使用时，即被创建并且赋值。是不是感觉很好奇，接下来看一下数据属性的使用。

首先来看一个小例子，代码如下：

```
class Datattribute:
    pass
if __name__=="__main__":
    data=Datattribute()
    data.name='我是没有被预先定义的数据属性'
    print data.name
```

在上述代码中，没有在 Datattribute 类中定义名称为 name 的属性。但在代码中可以直接使用，这就是数据属性的好处所在。其执行结果如下：

```
>>>
我是没有被预先定义的数据属性
>>>
```

4) 内置属性

在 Python 中，类提供了一些内置属性，用来管理类的内部关系。下面通过例子来看一下内置属性如何使用。代码如下：

```
class BuiltAttribute:
    def __init__ (self):
        self.built='wo shi fangfa "__init__"gou zao fang fa de shu xing'

class AttendBuilt(BuiltAttribute):
    def accept (self):
        self.acceptAttend='我是方法 accept 中的属性'
if __name__=="__main__":
    buildattribute=BuiltAttribute()
    attendbuilt=AttendBuilt()
    print '我是继承 BuiltAttribute 的属性',attendbuilt.built
    print '我是使用__bases__内置属性输出的基类组成的元祖',AttendBuilt.__bases__
    print '我是使用__dict__内置属性输出的属性组成的字典',attendbuilt.__dict__
    print '我是使用__module__内置属性输出的类所在的模块名',attendbuilt.__module__
    print '我是使用__doc__内置属性输出的 doc 文档',attendbuilt.__doc__
    print '我是使用__name__内置属性输出的类名',AttendBuilt.__name__
```

在上述代码中，使用内置属性__bases__来输出其父类组成的元祖。__dict__属性用来输出 attendbuilt 对象中属性组成的字典，__module__属性用来输出当前运行的模块名称，__doc__属性用来输出 doc 文档，而__name__属性用来输出当前对象的类名。下面来看一下执行的结果。

```
>>>
我是继承 BuiltAttribute 的属性 wo shi fangfa "__init__"gou zao fang fa de shu xing
我是使用__bases__内置属性输出的基类组成的元祖 (<class __main__.BuiltAttribute at
0x00CD4390>,)
我是使用__dict__内置属性输出的属性组成的字典 {'built': 'wo shi fangfa "__init__"gou
zao fang fa de shu xing'}
我是使用__module__内置属性输出的类所在的模块名 __main__
我是使用__doc__内置属性输出的 doc 文档 None
我是使用__name__内置属性输出的类名 AttendBuilt
```

```
>>>
```

2. 类的方法

通过对属性的学习，我们了解到属性分为公有属性和私有属性，同样方法也可以有公有方法和私有方法，其定义的规则是相同的。在 Java 语言中，使用关键字 static 来指定某方法是否为静态方法，但在 Python 语言中并没有使用 static 保留字，而是使用函数 staticmethod()或者通过@staticmethod 指令的方式来定义静态方法。

1) 静态方法

下面通过一个例子来说明静态方法的使用。代码如下：

```
class Methods:
    @staticmethod
    def mymethod ():
        print '我是被定义的静态方法'
    def __mymehtod ():
        print '我是私有的方法'
    def getMymethod ():
        print '我是测试转换为静态的方法'
    conversion =staticmethod(getMymethod)
    conPrivate =staticmethod(__mymehtod)
if __name__=="__main__":
    methods=Methods()                  #实例化对象
    methods.mymethod()                 #实体对象访问普通方法
    Methods.mymethod()                 #类访问普通方法
    Methods.conversion()               #类访问转换为静态后的普通方法
    methods.conversion()               #实体对象访问转换为静态后的普通方法
    Methods.conPrivate()               #类访问转换为静态的私有方法
    methods.conPrivate()               #实例对象访问转换为静态的私有方法
```

在上述代码中，在类 Methods 中分别声明了一个静态的方法 mymethod，一个私有方法__mymethod 和一个普通的方法 getMymethod。接着使用函数 staticmethod()将普通方法 getMymethod 转换为静态方法 conversion，将私有方法__mymethod 转换为静态方法 conPrivate。最后调用方法，执行结果如下：

```
>>>
我是被定义的静态方法
我是被定义的静态方法
我是测试转换为静态的方法
我是测试转换为静态的方法
我是私有的方法
我是私有的方法
>>>
```

从上述结果来看，在静态方法中不仅可以使用类，还可以使用实例化对象来访问。但是不能直接访问私有方法，否则会出现异常。

上面了解了静态方法的声明和使用，另外 Python 还提供了类方法和类实例方法，下面我们就来看一下这两种方法的使用和区别。

2) 类方法和类实例方法

类方法使用@classmethod 指令来声明，而实例方法则无须使用指令来声明。接下来通过一

个例子来说明。代码如下：

```
class MySl(object):
    def myfoo(self,myself):
    #类实例方法
        print "执行实例方法 foo(%s,%s)"%(self,myself)
    @classmethod
    def class_foo(cls,mycls):
    #类方法
        print "执行类方法 class_foo(%s,%s)"%(cls,mycls)
if __name__=="__main__":
     mysl = MySl()
     mysl.myfoo('dcy')
     mysl.class_foo('dcy')
     MySl.class_foo('dcy')
```

在上述代码中，先声明了一个类实例方法 myfoo 和一个类方法 class_foo，然后分别使用类和类实例调用类方法，而对类实例方法 myfoo 来说，只能使用类实例调用，否则会出现异常 TypeError。运行程序，执行结果如下：

```
>>>
执行实例方法 foo(<__main__.MySl object at 0x011A1070>,dcy)
执行类方法 class_foo(<class '__main__.MySl'>,dcy)
执行类方法 class_foo(<class '__main__.MySl'>,dcy)
>>>
```

上述结果表明，类方法隐含调用的参数是类，而类实例方法隐含调用的参数是类的实例。



在 Python 中，静态方法和类方法都可以被类和类实例调用，而类实例方法只能被类实例调用。

3. 内置方法

在前面的章节中，我们使用了__init__方法，你是不是对它感到很迷惑，不知道它和普通方法有什么区别？别急，学了这一节，你就会恍然大悟。

在 Python 语言中，类可以定义专用方法。也就是说，在特殊情况下或者当使用特别语法时由 Python 替你调用，这些方法有一个好听的名字就是内置方法。表 7-1 列出了比较常用的内置方法。

表 7-1　类的内置方法

内置方法	说　明
__init__(self,...)	初始化对象，在创建新对象前声明
__del__(self)	释放对象，在对象被删除之前调用
__new__(self,*args,**ked)	实例的生成操作
__str__(self)	在使用 print 语句时被调用
__delitem__(self,key)	从字典中删除 key 对应的元素
__setitem__(self,key,value)	为字典中的 key 赋值
__getitem__(self,key)	获取序列的索引 key 对应的值，等价于 seq[key]

__len__(self)	在调用内联函数 len()时被调用
__cmp__(src,dst)	比较两个对象 src 和 dst
__getattr__(s,name)	获取属性的值
__setattr__(s,name)	设置属性的值
__delattr__(s,name)	删除 name 属性
__gt__(self,other)	判断 self 对象是否大于 other 对象
__lt__(self,other)	判断 self 对象是否小于 other 对象
__ge__(self,other)	判断 self 对象是否大于或者等于 other 对象
__le__(self,other)	判断 self 对象是否小于或者等于 other 对象
__eq__(self,other)	判断 self 对象是否等于 other 对象
__call__(self,*args)	把实例对象作为函数调用

接下来，我们将针对表 7-1 中的几个比较重要的内置方法进行详细讲解。首先来看一下__init__方法。

1) __init__方法

想必大家都听说过构造函数这个词。在 Java 中，与类同名的函数方能称为方法。在 Python 中，同样也有构造函数，而__init__方法就是 Python 中的构造函数，在程序中起到初始化对象的作用。接下来我们就来看一下__init__方法的具体使用情况。

下面来看一个小例子，代码如下：

```
class Person:
    def __init__(self, name):
        self.name = name
    def sayHi(self):
        print 'Hello, my name is', self.name
p = Person('duanchunyang')
p.sayHi()
```

在上述代码中，在__init__方法中定义了两个参数，分别是 self 和 name，接着创建了一个新的域 self.name，然后创建一个 p 实例并传入参数。其执行结果如下：

```
Hello, my name is duanchunyang
```

是不是很奇怪？在上述代码中，并没有调用__init__方法，为什么会打印输出？原来在创建一个类的新实例时，参数被包括在圆括号内跟在类名后面，从而被传递给__init__方法。

下面来看__del__方法。

2) __del__方法

__del__方法的主要作用是释放被占用的资源，在 Python 中是析构函数。

下面通过一个例子来说明__del__方法的使用，代码如下：

```
class Room:
    count=0
    def __init__ (self,name):
        self.name=name
        print '初始化，传入的名称是%s'%self.name
        Room.count+=1
```

```
    def __del__ (self):
        print '%s 说 byebye'%self.name
        Room.count-=1
        if Room.count==0:
            print '我是这房间里的最后一个了'
        else:
            print '这间房子里有 %d 个人还未离开'%Room.count
    def sayHi (self):
        print '大家好,我的名字是 %s'%self.name
    def howMany (self):
        if Room.count==1:
            print '这个房间里就剩下我一个人了'
        else:
            print '这个房间里还有 %d 个人'%Room.count
if __name__=='__main__':
        room=Room('duanchunyang')
        room.sayHi()
        room.howMany()
        room1 = Room('lintiantian')
        room1.sayHi()
        room1.howMany()
        room.__del__()
```

在上述代码中，在初始化__init__方法时声明了两个变量 name 和 count，其中 name 是对象的变量，而 count 是类的变量。在该方法中，房间中每增加一个人，类变量 count 就需要加 1，而 self.name 的值根据传入的对象来指定。在 Room 类中，我们还使用了一个__del__方法，由于__del__方法很难保证其在什么时候运行，因此需要明确指明。接下来看一下执行的结果。

```
>>>
初始化，传入的名称是 duanchunyang
大家好,我的名字是 duanchunyang
这个房间里就剩下我一个人了
初始化，传入的名称是 lintiantian
大家好,我的名字是 lintiantian
这个房间里还有 2 个人
duanchunyang 说 byebye
这间房子里有 1 个人还未离开
>>>
```

下面来看一下__new__方法的使用。

3) __new__方法

__new__方法在创建对象时被调用，返回当前对象的一个实例。乍一看，是不是有些迷惑，感觉该方法和__init__方法的使用没有什么区别。实际上，__init__方法在创建完对象之后才被调用，对当前对象的实例进行初始化，而__new__方法则是在创建对象时被调用。接下来通过一个小例子来说明，代码如下：

```
class Mynew(object):
    def __init__(self):
        print ("__init__")
    def __new__(self):
        print ("__new__")
if __name__=='__main__':
    mynew=Mynew()
    mynew
```

在上述代码中，在类 Mynew 中使用了两个方法：__init__和__new__。那么程序执行的结果会如何呢？

```
__new__
```

如上述结果所示，结果打印出__new__，原来在类中默认创建的是__new__方法，然后调用__init__方法。在该例中我们重写了__new__方法，并且在该方法中没有调用__init__方法，因此__init__方法没有起任何作用。

4) __setitem__方法

__setitem__专用方法的含义是简单地重定向到真正的字典 self.data，让它来进行工作。下面看一个使用__setitem__方法的小例子，代码如下：

```
class MySetitem:
    def __setitem__(self, key, value):
        print 'key=%s,value=%s' % (key, value)
mysetitem = MySetitem()
mysetitem['dcy'] = 'duanchunyang'
mysetitem['mxl'] = 'maxianglin'
mysetitem['ltt'] = 'lintiantian'
```

在上述代码中，先创建了一个实例对象 mysetitem，然后将传入的 dcy 和 duanchunyang 分别对应__setitem__方法中的参数 key 和 value。具体执行结果如下：

```
key=dcy,value=duanchunyang
key=mxl,value=maxianglin
key=ltt,value=lintiantian
```

5) __getitem__方法

了解了__setitem__方法的使用，下面来看一下__getitem__方法在程序中的使用方法。专用方法__getitem__的含义是重定向到字典，返回的是字典的值，代码如下：

```
class MyGetitem:
    def __getitem__(self, key):
        if key == 'Thank you':
                return 'You are welcome! '
        elif key == 'Sorry':
                return  'That is all right'
        elif key == 'Do you like me':
                return 'I love you'
        else:
                return 'you can go'
if __name__=='__main__':
    mygetitem = MyGetitem()
    print mygetitem['Thank you']
    print mygetitem['Sorry']
    print mygetitem['Do you like me']
    print mygetitem['']
```

运行程序，执行结果如下：

```
>>>
You are welcome!
That is all right
I love you
you can go
>>>
```

6) __delitem__方法

__delitem__方法是在调用“del | 实例对象[key]”时调用，代码如下：

```
class MyDelitem:
    def __delitem__(self, key):
        print 'delete item:%s' %key
mydelitem = MyDelitem()
del mydelitem['dcy']
```

运行程序，其执行结果如下：

```
delete item:dcy
```

如上述结果所示，当在实例 mydelitem 中使用 del 方法时，Python 类会自动调用__delitem__方法。

7) __cmp__方法

在表 7-1 中提到__cmp__方法是用来比较两个实例对象。当使用==比较类实例时，Python 类将会替你调用__cmp__专用方法。下面通过例子来看一下其具体的使用情况。代码如下：

```
class MyCmp:
    def __cmp__(self,other):
        print '__cmp__ is called'
        return 0
if __name__=='__main__':
    mycmp1=MyCmp()
    mycmp2=MyCmp()
    print mycmp1==mycmp2
```

在上述代码中，声明了两个实例对象 mycmp1 和 mycmp2，当比较这两个对象时，会打印输出__cmp__ is called。接下来看一下执行结果。

```
__cmp__ is called
True
```

从上述结果可以看出，当比较的两个对象为真时，__cmp__专用方法的返回值是 0。

4．方法的动态特性

Python 语言是一种完全面向对象的动态语言，作为动态语言的脚本，用其编写的程序也具有很强的动态特性。主要体现在：可以动态添加类的方法，将某个已经定义的方法添加到类中。接下来看一下其语法格式。

```
class_name.method_name=exist_name
```

在上述语法中，class_name 指的是类名，method_name 指的是新的方法名，exist_name 指的是已经存在的方法名。接下来通过一个小例子来说明。

```
class Yesterday:
    pass
def today (self):
    print '我要牢牢把握今天，绝不虚度'
if __name__=='__main__':
    Yesterday.yesterday=today
    yes=Yesterday()
    yes.yesterday()
```

在上述代码中，定义了一个类 Yesterday，该类的内容由 pass 占位符代替。之后又定义了一个新的方法名称为 today，接着使用 Yesterday.yesterday=today 的方式将新的方法添加到 Yesterday 类中，并在该类中指定方法名为 yesterday。下面来看一下输出的结果。

```
>>>
我要牢牢把握今天，绝不虚度
>>>
```

从上述结果可以看出，已经将新创建的方法 today 添加到类 Yesterday 中。这样只需要一句代码就能将类外面的方法添加到类中，是不是很方便！

你可能会疑惑，上面这段代码使用 pass 占位符代替了类中的方法，如果在该类中存在重新定义的方法，那么结果会抛出异常还是会替换该方法中的内容呢？下面通过一个例子来求证一下，代码如下：

```
def among ():
    print '衣带渐宽终不悔，为伊消得人憔悴。'
def ending ():
    print '众里寻他千百度，蓦然回首，那人却在，灯火阑珊处'
class Person:
    def began (self):
        print '昨夜西风凋碧树。独上高楼，望尽天涯路。'
if __name__=='__main__':
    print '人生的三个境界,第一个境界是：'
    person=Person()
    person.began()
    print '第二个境界是：'
    person.began=among
    person.began()
    print '第三个境界是：'
    person.began=ending
    person.began()
```

在上述代码中，在 Person 类的内部定义了一个名称为 began 的方法，在类的外部定义了两个方法 among 和 ending，接着使用 person.began=among 的方式将新方法 among 添加到类 Person 中，并且在该类中的方法名称也是 began。最后使用 Person 类的实例 person 调用 began 方法，其执行结果如下：

```
>>>
人生的三个境界,第一个境界是：
昨夜西风凋碧树。独上高楼，望尽天涯路。
第二个境界是：
衣带渐宽终不悔，为伊消得人憔悴。
第三个境界是：
众里寻他千百度，蓦然回首，那人却在，灯火阑珊处。
>>>
```

从上述结果可以看出，在调用实例对象的 began 方法时，新定义方法的内容将原来类中方法的内容代替了。

7.3.2 实例描述

小时候看课本，记得有这样一个故事：牛顿被苹果砸到了头，因此发现了万有引力。碰巧我也有过被苹果砸到的经历，遗憾的是我什么都没发现。但换个角度一想，牛顿是伟大的天文学家，如果任何一个人都能发现其中的道理，那么“闻道有先后，术业有专攻”这句话就没有任何意义了。虽然我对天文地理一窍不通，但是我会编程，可以使用不同的编程语言将万物的生长过程展示出来，这就是我的优点。既然无法在天文或者地理方面发现问题，那就通过编程来描述一下苹果成熟的过程，借此来抚慰一下自己的心。

7.3.3 实例应用

【例 7-2】模拟水果成熟的过程。

(1) 创建一个名称为 apple.py 的文件。

(2) 在文件 apple.py 中添加代码。

```
class Fruit:
    def __init__(self, *args):
        for arg in args:
            arg(self)
    def config(self, *args):
        for arg in args:
            arg(self)

#是否成熟
def has_harvest(self):
    self.harvest = True
def has_not_harvest(self):
    self.harvest = False

#水果的颜色
def setColor(color):
    def method(self):
        self.color = color
    return method
#水果是否能吃
def can_eat(self):
    self.eat = True
def can_notEat (self):
    self.eat=False
if __name__=='__main__':
    apple = Fruit(has_not_harvest, setColor('green'))
    print '苹果是否成熟:%s;目前苹果的颜色:%s' % (apple.harvest, apple.color)
    apple.config(has_harvest, setColor('red'),can_notEat)
    print '苹果是否成熟:%s;目前苹果的颜色:%s;可以摘下来吃吗: %s' % (apple.harvest,
apple.color, apple.eat)
    apple.config(has_harvest, setColor('red'), can_eat)
    print '苹果是否成熟:%s;目前苹果的颜色:%s;可以摘下来吃吗:%s' % (apple.harvest,
apple.color, apple.eat)
```

(3) 保存修改好的代码。

7.3.4　运行结果

执行程序，运行结果如下：

```
>>>
苹果是否成熟:False;目前苹果的颜色:green
苹果是否成熟:True;目前苹果的颜色:red;可以摘下来吃吗: False
苹果是否成熟:True;目前苹果的颜色:red;可以摘下来吃吗:True
>>>
```

7.3.5　实例分析



源码解析

在本实例中，声明了__init__构造方法和 config 方法，并以*arg 为参数，意思是可以接收多个参数。另外还声明了水果是否成熟、水果的颜色、水果是否能吃等方法，最后根据传入的参数来创建类的实例，并调用 config 方法进行打印输出。

7.4　创建独特的服装连锁店

谈及继承，我想大家都不陌生。在老家最常听到的话就是“你看这小孩子长的多像他的爸爸妈妈呀，漂亮极了”，其中不免有遗传基因的存在。如果你够细心，就很容易发现，世界上的一切对象之间都有关系。例如，鼎鼎有名的服装连锁店“美特斯邦威”，它在全国各地都有分店，而分店中所有的衣服都是一个品牌，该分店相对于总店来说难道不可以认为是子类吗？答案是肯定的。如果你有能力、够聪明，为何不能使用程序建立一个独特的品牌连锁店呢。

 **视频教学：** 光盘/videos/07/继承.avi　　　　**长度：** 10 分钟

7.4.1　基础知识——继承

继承是指一个对象直接使用另一对象的属性和方法，它存在于面向对象程序设计中的类之间，是面向对象程序设计的重要手段。通过继承可以更有效地组织程序结构，明确类之间的关系，充分利用已有的类来完成更复杂、更深入的开发。采用继承的方式来组织设计系统中的类，还可以提高程序的抽象程度，更接近人的思维方式，使程序结构更清晰并降低编码和维护的工作量。

当一个类拥有另一个类的所有数据和操作时，就称这两个类之间具有继承关系。被继承的类称为父类或超类，继承了父类或超类的所有数据和操作的类称为子类。在 Java 语言中，可以用 extends 表明子类与父类的继承关系，那么在 Python 语言中如何使用继承呢？接下来将做详细介绍。

1. 使用继承

前面提到继承是直接使用另一对象的属性和方法。那么在 Python 中使用继承的语法是什么样呢？首先来看一下继承的语法。

```
class class_name(fatherclass_name)
```

在上述语法中，fatherclass_name 代表的是 class_name 类要继承的类。

例如：在学校里，有老师和学生，他们都有姓名、年龄、地址和爱好等特性，不同的是老师有工资，而学生没工资只有学分。可以将学校所有的成员作为一个共同的类，使老师和学生继承这个共同的类，也就是说老师和学生是该共同类的子类，之后分别为这些子类添加专有的属性。其具体实现的代码如下：

```
class SchoolMember:
    def __init__ (self,name,age,addr,hoppy):
        self.name=name
        self.age=age
        self.addr=addr
        self.hoppy=hoppy
        print '初始化的名字是%s'%self.name

    def tell (self):
        print '姓名:%s,年龄:%s,地址: %s,爱好:
%s'%(self.name,self.age,self.addr,self.hoppy)

class Teacher(SchoolMember):
    def __init__ (self,name,age,addr,hoppy,salary):
        SchoolMember.__init__(self,name,age,addr,hoppy)
        self.salary=salary
        print '继承 SchoolMember 传入的名字:%s'%self.name

    def tell (self):
        SchoolMember.tell(self)
        print '我的工资一般是:%s'%self.salary

class Student(SchoolMember):
    def __init__ (self,name,age,addr,hoppy,marks):
        SchoolMember.__init__(self,name,age,addr,hoppy)
        self.marks=marks
        print '我是继承 SchoolMember 学生%s'%self.name

    def tell (self):
        SchoolMember.tell(self)
        print '我这次的成绩是:%d'%self.marks

t=Teacher('dcy',40,'河南郑州','旅游',3000)
s=Student('mxl',22,'上海','导游',85)
members=[t,s]
for member in members :
    member.tell()
```

在上述代码中，创建了一个共同的类 SchoolMember，在该类中声明了共同的属性 self、name、age、addr 和 hoppy，接着创建了 Teacher 和 Student 两个类，分别继承 SchoolMember 类。在 Teacher 类中添加了属性 salary，在 Student 类中添加了属性 marks，最后分别创建 Teacher

和 Student 的实例，然后将实例存放到一个元组中，并使用 for 循环语句进行打印输出。执行结果如下：

```
>>>
初始化的名字是 dcy
继承 SchoolMember 传入的名字:dcy
初始化的名字是 mxl
我是继承 SchoolMember 学生 mxl
姓名:dcy,年龄:40,地址：河南郑州,爱好：旅游
我的工资一般是:3000
姓名:mxl,年龄:22,地址：上海,爱好：导游
我这次的成绩是:85
>>>
```

说到继承，必然会提到如何调用父类中的方法，一般使用非绑定的类方法，即使用类名访问父类中的方法，并在参数列表中引入对象 self，从而达到调用父类的目的。下面来看一段代码。

```
class Father:
  def __init__(self):
   print "我是初始化 Father 类中的方法"
   print "供以后调用"
class Son(Father):
  def __init__(self):
   print "我是初始化 Son 类中的方法"
   Father.__init__(self)
b = Son()
```

在上述代码中，使用 Father.的方式来访问 Father 类中的__init__方法，执行结果如下：

```
>>>
我是初始化 Son 类中的方法
我是初始化 Father 类中的方法
供以后调用
>>>
```

不知道你有没有发现：如果 Son 的父类由 Father 变为 ExtendFather，那么就需要遍历整个类定义，将所有的类名替换过来，请看下述代码：

```
class Son(ExtendFather):
  def __init__(self):
   print "我是初始化 Son 类中的方法"
   ExtendFather.__init__(self)
b = Son()
```

如果是简短的代码，这样的改动还可以接受；如果代码量很大，这样的修改会不会造成灾难性的故障呢？为了解决这个问题，Python 添加了一个关键字 Super 来调用父类中的方法。下面通过一段代码来了解 Super 的使用。

```
class Person(object):
    def __init__ (self,name):
        self.name=name
        print '我是一个人,初始化的名字是：%s'%self.name
class Star(Person):
    def __init__ (self,name):
        super(Star,self).__init__(name)
```

```
        print '我是一个大明星'
if __name__=='__main__':
    Star('dcy')
```

在上述代码中，如果将 Star 的父类修改为 PersonStar，那么还需要修改的 Star 类代码。

```
class Star(PersonStar):
    def __init__ (self,name):
        super(Star,self).__init__(name)
        print '我是一个大明星'
```

其执行结果是相同的，如下所示：

```
>>>
我是一个人,初始化的名字是：dcy
我是一个大明星
>>>
```

使用 Super 关键字来调用父类中的方法，倘若父类名称改变，则只需要修改子类继承父类的名称即可。这样既可以将代码的维护量降到最低，又可以提高程序开发的周期，很方便。

2. 抽象类的模拟

所谓抽象类，即是对一类事物的特征和行为的抽象，由抽象方法组成。在 Java 中，可以使用 abstract 来表示抽象类，但在 Python 2.5 中，并没有提供抽象类的语法，这是不是就意味着在 Python 语言中无法形容抽象类呢？当然不是，因为我们可以模拟抽象类。由于抽象类不能被实例化，因此可以通过 Python 中的 NotImplementedError 类来模拟抽象类，使其在实例化时抛出异常。这里的 NotImplementedError 类继承自 Python 运行时错误类 RuntimeError。

下面来看一个模拟抽象类的例子，代码如下：

```
def abstract():
   raise NotImplementedError("abstract")
class Person:
   def __init__ (self):
      if self.__class__ is Person:
         abstract()
class Star(Person):
    def __init__ (self):
        Person.__init__(self)
        print '我是一个大明星'
if __name__=='__main__':
    star=Star()
```

在上述代码中，定义了一个抽象方法 abstract，使之抛出 NotImplementedError 异常。接着定义了一个 Person 类，在该类中调用已定义的抽象方法，之后又定义了一个 Star 类，其继承自 Person 类，最后将类 Star 实例化为 star。运行程序，执行结果如下：

```
>>>
我是一个大明星
>>>
```

要将 Person 类也实例化，代码如下：

```
    person=Person()
```

再次运行程序，执行结果如下：

```
>>>
我是一个大明星
Traceback (most recent call last):
  File "F:\Python\11.py", line 13, in <module>
    person=Person()
  File "F:\Python\11.py", line 6, in __init__
    abstract()
  File "F:\Python\11.py", line 2, in abstract
    raise NotImplementedError("abstract")
NotImplementedError: abstract
>>>
```

从上述结果可以看出，如果实例化 Person 类，那么就会抛出之前定义的 abstract 异常，这说明模拟的抽象类成功了。

3. 多重继承

想必你已经发现，上面介绍的继承只有一个父类，但是如果有多个父类，那么子类该如何继承呢？别急，Python 提供了多重继承的方法，其语法格式如下：

```
class class_name(fatherclass_name,fatherclass_name1,...)
```

在上述语法中，class_name 指的是类名，而 fatherclass_name 和 fatherclass_name 指的是父类名。例如：在很小的时候，邻居们见了我，总是对我的容貌各抒己见。有的说：你看她的眼睛真想她爸爸，都是双眼皮；有的说：她的额头有点宽，和她妈妈的一样；有的说：我的鼻子和姑姑的鼻子挺像的，高鼻梁。我想这或许就和程序上的多重继承类似吧。下面就以此为例，练习一下多重继承的使用，代码如下：

```
class MyFather:
    def __init__ (self):
        self.eyes='爸爸的眼睛是双眼皮'
        print self.eyes
class MyMother:
    def __init__ (self):
        self.forehead='妈妈的额头有点宽'
        print self.forehead
class MyAunt:
    def __init__ (self):
        self.nose='姑姑的鼻子是高鼻梁'
        print self.nose
class MySelf(MyFather,MyMother,MyAunt):
    def __init__ (self,face):
        print '我的眼睛是双眼皮，别人说我继承的是：'
        MyFather.__init__(self)
        print '我的额头有点宽，别人说我继承的是：'
        MyMother.__init__(self)
        print '我的鼻子有点高，还有人说我继承的是：'
        MyAunt.__init__(self)
        self.face=face
        print '我的脸型：%s'%self.face,'这下终于没人说我像谁了'
myself=MySelf('偏圆吧')
```

运行程序，执行结果如下：

```
>>>
我的眼睛是双眼皮，别人说我继承的是：
```

```
爸爸的眼睛是双眼皮
我的额头有点宽，别人说我继承的是：
妈妈的额头有点宽
我的鼻子有点高，还有人说我继承的是：
姑姑的鼻子是高鼻梁
我的脸型：偏圆吧 这下终于没人说我像谁了
>>>
```

7.4.2 实例描述

我想每个人都有自己的梦想。虽说梦想实现的成功率很小，但是我们有渴望梦想的权利。我的梦想就是开一个大型的服装店，在这个服装店里有各式各样的服装，并且在全国各地都有分店，其中分店与总店的不同在于规模的大小。但店中的服装都是同一品牌，其知名度类似于真维斯、暖倍儿。我的梦想是不是很美呀？嘘，别打扰我！让我在程序中把这个美梦做完。

7.4.3 实例应用

【例 7-3】创建独特的服装连锁店。

(1) 创建一个名称为 extend.py 的文件。

(2) 在文件 extend.py 文件中添加代码。

```
def abstract():
   raise NotImplementedError("对不起，不允许实例化超类")
class MyClothStore:
    #将该服装店的名称初始化
    def __init__(self):
        self.fname = 'SIMILE'
        print self.fname
        if self.__class__ is MyClothStore:
            abstract()
class MyGrilCloth(MyClothStore):
    def __init__ (self):
        self.clothname='甜美可私服'
        print self.fname
class MyBoyCloth(MyClothStore):
    def __init__ (self):
        self.clothname='太可思西服'
        print self.clothname
class BoyCloth(MyClothStore,MyBoyCloth):
    def __init__ (self,AdultName,AdultMake,AdultPrice,AdultWash):
        print '这件男装在全国的总店名称是：'
        MyClothStore.__init__(self)
        print '这件男装的名称是：'
        MyBoyCloth.__init__(self)
        self.AdultName=AdultName
        print '这件衣服的名称是:%s'%self.AdultName
        self.AdultMake=AdultMake
        print '这件衣服的制造是:%s'%self.AdultMake
        self.AdultPrice=AdultPrice
```

```
        print '这件衣服的价格是:%s'%self.AdultPrice
        self.AdultWash=AdultWash
        print '这件衣服只能被是:%s'%self.AdultWash
class AdultCloth(MyClothStore,MyGrilCloth):
    def __init__ (self,AdultName,AdultMake,AdultPrice,AdultWash):
        print '该女士服装在全国的总店名称是：'
        MyClothStore.__init__(self)
        print '该女士服装的名称是：'
        MyGrilCloth.__init__(self)
        self.AdultName=AdultName
        print '这件衣服的名称是:%s'%self.AdultName
        self.AdultMake=AdultMake
        print '这件衣服的制造是:%s'%self.AdultMake
        self.AdultPrice=AdultPrice
        print '这件衣服的价格是:%s'%self.AdultPrice
        self.AdultWash=AdultWash
        print '这件衣服只能被是:%s'%self.AdultWash
if __name__=='__main__':
     adultcloth=AdultCloth('dcy','guochan','1500RMB','干洗')
     adultcloth=BoyCloth('taikesi','guochan','2500RMB','干洗')
```

(3) 保存修改的代码。

7.4.4　运行结果

运行程序，执行结果如下：

```
>>>
该女士服装在全国的总店名称是：
SIMILE
该女士服装的名称是：
SIMILE
这件衣服的名称是:dcy
这件衣服的制造是:guochan
这件衣服的价格是:1500RMB
这件衣服只能被是:干洗
这件男装在全国的总店名称是：
SIMILE
这件男装的名称是：
太可思西服
这件衣服的名称是:taikesi
这件衣服的制造是:guochan
这件衣服的价格是:2500RMB
这件衣服只能被是:干洗
>>>
```

7.4.5　实例分析



源码解析

在本实例中，先定义了一个抽象方法 abstract()，接着创建了 MyClothStore、MyGrilCloth、MyBoyCloth、MyBoyCloth 和 AdultCloth 五个类，其中 AdultCloth 类继承自 MyClothStore 和 MyGrilCloth 类，而 MyBoyCloth 类继承自 MyClothStore 和 MyBoyCloth 类。如果你想要实例化 MyClothStore 类，就会引发在抽象方法 abstract()中定义的 NotImplementedError 异常。

7.5　类的其他特性

在前面的章节中，我们已经了解了 Python 类的方法、属性以及方法的动态特性。接下来将介绍类的其他特性，例如类命名空间的使用，如何判断类是否继承另外一个类以及对象是不是某个类的实例。首先介绍类的命名空间。

 **视频教学**：光盘/videos/07/类的命名空间.avi　　　　长度：7 分钟

7.5.1　基础知识——类的命名空间

在程序开发过程中，类和类成员的名称是丰富的，为了描述一个具体的对象，需要对类和类成员进行设计。在设计类和类成员的过程中，难免会出现类的名称或者类成员中的方法相同的情况，这样就会造成代码混乱，从而使代码的可读性降低。使用命名空间可以解决此类问题。

在 Java 中，我们只需要把类定义放到各自的包中，就可以避免类的名称或者类成员重复了。我们可以认为 Package 是 Java 中的命名空间，而在 C++中的命名空间是使用 namespace 关键字来命名，在该命名空间里不仅可以包括类，还可以包括函数、变量、模板等。那么在 Python 语言中的命名空间是怎样的呢？接下来我们就来看一下。

在 Python 语言中定义类时，所有位于 class 语句中的代码都在特殊的命名空间中执行，该命名空间被称为类命名空间(class namespace)。这个类命名空间不仅可以被类中所有成员访问，还可以被类的实例方法访问。接下来我们通过一个小例子来说明类命名空间的使用，代码如下：

```
class MyNamespace:
    count=1
    def myinit (self):
        MyNamespace.count+=1
if __name__=='__main__':
    mynamespace=MyNamespace()
    mynamespace.myinit()
    print MyNamespace.count
```

在上述代码中，声明了一个类命名空间 MyNamespace，在 MyNamespace 中定义了一个可供所有成员访问的变量 count，用来计算成员的数量，紧接着定义了一个 myinit 方法来初始化所有的实例。之后将类实例化，调用方法 myinit 进行输出，执行结果如下：

```
>>>
2
>>>
```

7.5.2　基础知识——检查继承

在前面的章节中，我们了解了什么是继承以及如何使用继承，但是如何判断一个类是不是另一个类的子类呢？想必都会说：在类名后括号中的类就是基类(父类)，这只是从感官上来判断，如何证明呢？这就需要使用 Python 提供的内建 issubclass 函数了。下面就来看一下如何使用 issubclass 函数。

假如有两个类，分别是 Person 和 Ordinary，其代码如下：

```
class Person(object):
    def __init__ (self,name):
        self.name=name
        print '我是一个人,初始化的名字是：%s'%self.name

class Ordinary(Person):
    def __init__ (self):
        super(Ordinary,self).__init__()
        print '我是一个普通人'
```

在上述代码中，根据学习 Python 语言的经验，很容易看出 Ordinary 类继承自 Person 类，如果要证明 Ordinary 类是子类，需要添加如下代码：

```
if __name__=='__main__':
    print issubclass(Ordinary,Person)
```

在上述代码中，使用了 issubclass 函数，如果输出的结果为 True，那就说明 Ordinary 类是 Person 的子类，否则就不是。执行结果如下：

```
>>>
True
>>>
```

同样，也可以使用 isinstance 方法来检查一个对象是不是一个类的实例。接下来仍使用含有 Person 和 Ordinary 类的代码，我们只需要添加如下代码：

```
if __name__=='__main__':
    person=Person('dcy')
    print isinstance(person,Person)
```

在上面这段代码中，为类 Person 声明的实例名称为 person，接着通过 isinstance 方法判断 person 对象是否为 Person 类的实例，如果输出的结果为 True，则说明是，否则就不是。

7.6　新　式　类

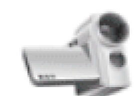
看到“新式(new-style)类”这个词是不是很疑惑，难道还有“旧式类”吗？答案是肯定的。只不过所谓的旧式类有一个好听的名字叫做经典(classic)类。在这里提到的新式类是在 Python 2.2 中引入的。

通常情况下，从 object 或者其他内置类型衍生的类，都被称为新式类(所谓衍生，指的是

包括 object 的子类、object 的子类的子类以及那些有内置类型且位于超类的某个地方的类，新的类就会被当作新式类)。而那些不是从内置类型衍生出来的类，就会被当作经典类来处理。

相对于经典类，新式类只是多了一些高级功能。其代码的编写也是使用之前学过的正常类的语法，主要差别就在于新式类是从内置类型创建的子类，如果没有合适的内置类型可以使用，那么新的内置名称 object 便作为新式类的超类。

接下来看一下新式类启用了那些高级特性，首先来看一下__slots__类属性。

 **视频教学：**光盘/videos/07/新式类.avi　　　　长度：9 分钟

7.6.1　基础知识——_slots_类属性

众所周知，Python 是一门动态语言，可以在运行过程中修改对象的属性和增删方法。任何类的实例对象都包含一个字典__dict__，Python 通过这个字典将任意属性绑定到对象上。由于字典会占据大量内存，如果你有一个属性数量很少的子类，但有很多的实例，从内存上考虑，我们可以使用__slots__属性来代替__dict__属性。

__slots__是一个类变量，可以由一系列对象组成，使用所有合法标识构成的实例属性的集合来表示。它也可以是一个列表、元组或可迭代对象。总之，任何试图创建一个其名不在__slots__中的实例属性的操作都将引发 AttributeError 异常。

一般情况下，__slots__类属性在 class 语句顶层设置。下面我们通过一个小例子来做说明，其代码如下：

```
class MyLimeter(object):
    __slots__='myname','myage','myhoppy'
if __name__=='__main__':
    x=MyLimeter()
    x.myname
```

在上述代码中，很明显就能看出使用的是新式类，因为类 MyLimeter 是以 object 类为超类的。在__slots__类属性中，我们定义了 3 个属性 myname、myage 和 myhoppy，之后创建了 MyLimeter 类实例，并调用属性 myname，执行结果如下：

```
>>>
Traceback (most recent call last):
  File "F:\Python\18.py", line 8, in <module>
    x.myname
AttributeError: myname
>>>
```

是不是很奇怪，为什么会引发异常呢？原来，__slots__类属性和其他 Python 中的变量名一样，在实例属性名被引用前，必须赋值。我们可以将代码 x.myname 修改如下：

```
x.job='白领'
```

运行程序，执行结果如下：

```
>>>
Traceback (most recent call last):
  File "F:\Python\18.py", line 5, in <module>
    x.job='白领'
```

```
AttributeError: 'MyLimeter' object has no attribute 'job'
>>>
```

这下你是不是彻底郁闷了：赋了值，还出现错误！别急，仔细看一下你就会发现错误之源。原来，__slots__类属性可以防止用户随心所欲地动态添加实例属性。再次将代码进行 x.job="白领"修改。

```
x.myname='duanchunyang'
```

保存代码，进行打印，执行结果如下：

```
>>>
duanchunyang
>>>
```

7.6.2　基础知识——__getattribute__()特殊方法

在 Python 类中有一个特殊的方法__getattr__()，它仅当属性不能在实例的__dict__、类的__dict__或者__dict__中找到时才被调用。大多数的情况是，我们想要一个适当的函数来执行对每个属性的访问，而不仅仅是属性找不到的情况，这就需要使用__getattribute__()方法了。

__getattribute__()方法只适用于新式类。下面我们通过一个例子来说明__getattribute__()方法的使用，代码如下：

```
class MyAttribute(object):
    def __init__(self):
        self.default = 0.0
        self.age = 20
        self.member = 21
    def __getattribute__(self, name):
        if name == 'test':
                print ("当调用的属性为 test 时，经过__getattribute__方法%s" %name)
                return self.test
        else:
                print ("当调用的属性不是 test 时，打印输出的值%s" % name)
                return object.__getattribute__(self, name)

    def __getattr__(self, name):
        print ("经过__getattr__方法，打印输出的 name 值是： %s" % name)
        print ("打印出的原先设置的 default 的值：%s" % self.default)
        return self.default
if __name__ == "__main__":
    myattribute = MyAttribute()
    myattribute.member
    myattribute.so
```

在上述代码中，在类 MyAttribute 中使用了__init__构造方法来初始化 default、age 和 member 属性。接着使用__getattribute__方法来确定是不是对每个属性的访问都需要调用该方法，并且在该方法中使用 if 语句判断调用的属性是否为 test，如果是则执行相应的代码块。之后又使用了__getattr__方法来确定所创建的实例调用的属性是否在类中。最后创建 MyAttribute 类的实例，分别访问 member 和 so 属性，执行结果如下：

```
>>>
```

```
当调用的属性不是 test 时所打印输出的值 member
当调用的属性不是 test 时所打印输出的值 so
经过__getattr__方法，打印输出的 name 值是： so
当调用的属性不是 test 时，打印输出的值 default
打印出的原先设置的 default 的值：0.0
>>>
```

从上述结果来看，当实例对象 myattribute 访问属性 member 时，调用__getattribute__方法，并且判断该属性是否为 test，很显然不是，则执行 else 代码块。由于属性 member 在类 MyAttribute 中存在，因此没有调用__getattr__方法中的内容。当实例对象 myattribute 访问属性 so 时，又一次调用__getattribute__方法，进入__getattribute__方法后进行判断并且执行相应的代码，由于属性 so 在 MyAttribute 类中不存在，因此调用__getattr__方法。在__getattr__方法中，当执行到 self.default 时，又调用了一次__getattribute__方法，接着再次判断，并且执行 else 块中的代码，最后进行打印输出。

你是不是很好奇，当调用的属性名为 test 时，会出现什么样的结果？很遗憾，执行结果会出现死循环。那么如何挽救这种状况呢？别着急，可以调用祖先类的同名方法，例如 object.__getattribute__(self, name)，这种方式只在新式类中有效。

7.6.3 基础知识——描述符

描述符是 Python 新式类中的关键符号之一。它为对象属性提供强大的 API，其原理就是将某种特殊类型的类的实例指派给另一个类的属性。这个特殊的描述符类是一种新式类，包含的方法有__get__()、__set__()和__delete__()(或者至少包含其中的几种)，其中__get__()方法用于得到一个属性的值，__set__()方法是为一个属性赋值，而__delete__()方法则是在采用 del 语句或者明确删除某个属性时被调用。

接下来我们来看一下__get__()、__set__()和__delete__()方法的原型。

```
def __get__(self, obj, typ=None)
def __set__(self, obj, val)
def __delete__(self, obj)
```

下面我们看一个关于描述符的例子，代码如下：

```
class MyDescription(object):
    def __get__(self, obj, typ=None):
        print '__get__方法, oby 是: %r, typ 是: %r' % (obj, typ)
        return self.data
    def __set__(self, obj, val):
        print '__set__方法, obj 是: %r, val 是: %r' % (obj, val)
        self.data=val
    def __delete__ (self,obj):
        print '__delete__方法, obj 是 %r'%obj
        del self.data

class MyDesClass(object):
    this = MyDescription()
    that = MyDescription()
    other=4
mydesclass = MyDesClass()
```

```
mydesclass.this = 'nishiwodeweiyi'
print mydesclass.this
print '我是打印出类 MyDesClass 中 other 的值：',mydesclass.other
mydesclass.other=6
print '我是打印出重新赋值给 other 的值：',mydesclass.other
```

在上述代码中，我们定义了类 MyDescription 和类 MyDesClass，其中 MyDesClass 类将 this 和 that 定义为 MyDescription 类的描述符，而属性 other 只是一个普通的类属性，然后将类 MyDesClass 实例化，接着对描述符和类属性分别进行操作。最后运行程序，执行结果如下：

```
>>>
__set__方法，obj 是： <__main__.MyDesClass object at 0x011A10B0>, val 是：
'nishiwodeweiyi'
__get__方法，oby 是： <__main__.MyDesClass object at 0x011A10B0>, typ 是： <class
'__main__.MyDesClass'>
nishiwodeweiyi
我是打印出类 MyDesClass 中 other 的值： 4
我是打印出重新赋值给 other 的值： 6
>>>
```

从上述结果来看，当我们第一次访问 mydesclass.other 时，程序将读出类属性。对其赋值后，将读取实例属性，此时类属性依然存在，只是被隐藏了。不信吗？添加下面的代码看看。

```
print '我是使用__class__访问类属性 other，结果是：',mydesclass.__class__.other
```

运行程序，执行的结果如下所示。

```
我是使用__class__访问类属性 other，结果是： 4
```

7.7　常见问题解答

7.7.1　Python中的__getattr__问题



Python 中的__getattr__问题。
网络课堂：http://bbs.itzcn.com/thread-12373-1-1.html

有这样一段代码，代码如下：

```
class Time:
   def __init__( self, hour = 0, minute = 0, second = 0 ):
      self.hour = hour
      self.minute = minute
      self.second = second
   def __setattr__( self, name, value ):
      if name == "hour":
         if 0 <= value < 24:
            self.__dict__[ "_hour" ] = value
         else:
            raise ValueError, "Invalid hour value: %d" % value
      elif name == "minute" or name == "second":
         if 0 <= value < 60:
            self.__dict__[ "_" + name ] = value
```

```
        else:
            raise ValueError, "Invalid %s value: %d" % \
                ( name, value )
    else:
        self.__dict__[ name ] = value
def __getattr__( self, name ):
    if name == "hour":
        return self._hour
    elif name == "minute":
        return self._minute
    elif name == "second":
        return self._second
    else:
        raise AttributeError, name
def __str__( self ):
    return "%.2d:%.2d:%.2d" % \
        ( self._hour, self._minute, self._second )
```

我知道__getattr__指的是在__dict__属性中找不到属性名时执行，在这个例子中，请问一下__getattr__在什么情况下被执行，在找不到哪个属性名时执行，很费解！希望大哥大姐帮忙，小弟感激不尽！

【解决办法】很高兴为你解答。

关于 Python 的内置方法，希望用一个简单的例子帮到你。代码如下：

```
class a(object):
    name = "Jim"
    def __setattr__(self,name,value):
        self.__dict__[name] = value
    def __getattr__(self,name):
        return name
b = a()
print '打印出的是 b 实例对象的目录：',dir(b)
print '打印出的是 b.__dict__对象：',b.__dict__
print '访问的是定义好的 name 属性的值：',b.name
print '------------------------------------'
b.name="change"
print '重新定义的 name 属性打印的目录：',dir(b)
print '打印出的是 b.__dict__对象：',b.__dict__
print '访问的是定义好的 name 属性的值：',b.name
```

运行程序，执行结果如下：

```
>>>
打印出的是 b 实例对象的目录： ['__class__', '__delattr__', '__dict__', '__doc__',
'__getattr__', '__getattribute__', '__hash__', '__init__', '__module__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__',
'__weakref__', 'name']
打印出的是 b.__dict__对象： {}
访问的是定义好的 name 属性的值： Jim
------------------------------------
重新定义的 name 属性打印的目录： ['__class__', '__delattr__', '__dict__', '__doc__',
'__getattr__', '__getattribute__', '__hash__', '__init__', '__module__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__str__',
'__weakref__', 'name']
打印出的是 b.__dict__对象： {'name': 'change'}
访问的是定义好的 name 属性的值： change
```

```
>>>
```

可以看到刚开始的时候 b 的__dict__对象里面是空的，当我们使用 b.name 访问的时候直接访问之前定义的 name 属性。

当我们使用 b.name="change"就会默认调用内置方法__setattr__，此时并没有直接操作 self.name 属性，而是在修改 self.__dict__['name']='change'。之后再用 b.name 访问的时候获得的就是__dict__中的 name 属性的值。

我们可以看到，__getattr__方法首先在__dict__中找属性名，然后在实例的属性中找，最后调用__getattr__方法获得值。

这样说，你能明白吧！

7.7.2　Python中的继承问题



Python 中的继承问题。
网络课堂：http://bbs.itzcn.com/thread-12373-1-1.html

在 Python 中，如何在继承类中调用基类的方法。

```
class father:
    def show(self):
        print "father"
class son(father):
    father.show(self)
```

执行结果出现错误，代码如下：

```
Traceback (most recent call last):
  File "fengzhuang.py", line 4, in <module>
    class son(father):
  File "fengzhuang.py", line 5, in son
    father.show(self)
NameError: name 'self' is not defined
```

这是怎么回事？

【解决办法】很高兴为你解答。

在你的代码中，为什么找不到构造器？你想拿什么来调用父类方法呢？你可以这样改写代码。

```
class Father:
    def __init__ (self):
        self.name='father'
class Son(Father):
    def __init__ (self):
        Father.__init__(self)
    def show (self):
        print 'son extends is %s'%self.name
son=Son()
son.show()
```

运行程序，执行结果如下：

```
son extends is father
```

7.7.3 Python中的__getattribute__问题



Python 中的__getattribute__问题。
网络课堂：http://bbs.itzcn.com/thread-12373-1-1.html

我使用__getattribute__方法来做了一个小例子，但程序在运行时却出现了死循环。我的代码如下：

```
class MyAttribute(object):
   def __init__(self):
      self.default = 0.0
      self.age = 20
      self.member = 21
   def __getattribute__(self, name):
      if name == 'dcy':
            print ("当调用的属性为 test 时，经过__getattribute__方法%s" %name)
            return self.dcy
      else:
            print ("当调用的属性不是 test 时所打印输出的值%s" % name)
            return object.__getattribute__(self, name)
if __name__ == "__main__":
   myattribute = MyAttribute()
   myattribute.member
   myattribute.dcy
```

这个问题困了我好几天了，请各位帮帮忙！

【解决办法】很高兴为你解答。

当属性被访问时，__getattribute__方法就会一直被调用，因此当实例对象 myattribute 访问属性 dcy 时，会调用__getattribute__方法，接着判断，之后又调用 self.dcy，因此会再次执行__getattribute__方法，从而造成死循环。我们只需要将 return self.dcy 这句代码修改为 return object.__getattribute__(self, name)就不会造成死循环了。你试一下吧。

7.8 习　　题

一、填空题

(1) 在 Java 语言中，我们使用访问修饰符 private 来表示该属性或者方法为私有，而在 Python 语言中，我们使用__________方式来表示该方法或者属性为私有。

(2) 在 Java 中，如果我们想将一个方法设置为静态方法，可以使用 static 关键字，而在 Python 语言中，我们使用 staticmethod()方法或者__________指令来设置。

(3) 在 Python 语言中，我们使用__________指令将该方法标注为类方法。

(4) 有下面一段代码。

```
def ending ():
    print '天下第二'
class Person:
    def began (self):
```

```
        print '天下第三'
if __name__=='__main__':
    person=Person()
    __________
    person.began()
```

如果我想将 ending()方法添加到类 Person 中，则需要在代码的空白处填写__________才能保证输出的结果是“天下第二”呢?

(5) 在 Python 语言中，我们使用__________函数来判断该类是否继承自另外一个类。

(6) 当我试图创建一个其名不在__________中的实例属性时将会引发 AttributeError 异常。

(7) 在 Python 语言中，当实例调用的属性找不到时，会调用__getattr__()方法；当实例调用的属性无论找到与否都会执行的方法是__________。

二、选择题

(1) 在下面的几个选项中，对私有属性 name 的声明正确的是__________。

A. __name　　　　B. private name

C. private string name　　　　D. private __name

(2) 在下面的代码中，定义静态方法正确的是__________。

A.

```
class Methods:
    staticmethod
    def mymethod ():
        print '我是被定义的静态方法'
```

B.

```
class Methods:
    @staticmethod
    def mymethod ():
        print '我是被定义的静态方法'
```

C.

```
class Methods:
    def staticmethod mymethod ():
        print '我是被定义的静态方法'
```

D.

```
class Methods:
    def @staticmethod mymethod ():
        print '我是被定义的静态方法'
```

(3) 有下面的代码。

```
class A:
    def __init__ (self):
        self.name='A'
        print self.name
class B:
    def __init__ (self):
        self.name='B'
        print self.name
class C:
```

```
    def __init__ (self):
        self.name='C'
        print self.name
```

如果有一个类 D，需要多重继承 A、B 和 C 三个类，那么下面几个选项中正确的是______。

A.

```
class D:
    def __init__ (self,face):
        print 'D'
        A.__init__(self)
        print 'A'
        B.__init__(self)
        print 'B'
        C.__init__(self)
        print 'C'
```

B.

```
class D extendA,B,C:
    def __init__ (self,face):
        print 'D'
        A.__init__(self)
        print 'A'
        B.__init__(self)
        print 'B'
        C.__init__(self)
        print 'C'
```

C.

```
class D(A,B,C):
    def __init__ (self,face):
        print 'D'
        A.__init__(self)
        print 'A'
        B.__init__(self)
        print 'B'
        C.__init__(self)
        print 'C'
```

D.

```
class D extends(A,B,C):
    def __init__ (self,face):
        print 'D'
        A.__init__(self)
        print 'A'
        B.__init__(self)
        print 'B'
        C.__init__(self)
        print 'C'
```

(4) 例如，我创建了一个 world 类，代码如下：

```
class World:
    def __init__ (self):
        print '我的世界我做主'
```

另外，我又添加了一个 sky 类，其代码如下：

```
class Sky(world):
    def __init__ (self):
        print '这个世界总有我自己的一片天空，努力'
```

接下来我使用 issubclass 函数判断该 sky 类是不是 world 的子类，代码如下：

```
if __name__=='__main__':
    print issubclass(Sky, World)
```

下面几个选项中，正确的是__________。

A. 0　　B. 1　　C. False　　D. True

三、上机练习

上机练习：继承和多态的使用。

本章主要介绍的是面向对象编程，其中面向对象最基本的三大要素就是封装、继承和多态。在程序中使用封装、继承和多态的好处就是可以使类具有独立性、通用性和灵活性，继承和多态又是紧密相连的，本次上机练习主要针对多态和继承。

例如：在每逢过年的时候，家里都会煮肉。煮鸡，煮鸭，杀牛宰羊，不亦乐乎。总的来说，这些都属于动物。我们可以动物为父类，在父类中创建这些动物具有共同特征的方法，以鸡、鸭为子类继承该父类。接着创建一个人的类，在该类中建立一个方法用来接收动物对象。当传入的对象是使用鸡的类创建的实例时，执行结果如下：

```
>>>
我的名字叫：Kobe
这只火鸡已经被吃了
>>>
```

当传入的对象是使用狗的对象创建的实例时，执行结果如下：

```
>>>
我的名字叫：Kobe
这狗肉已经被吃了
>>>
```



第 8 章　基于文件的交互

内容摘要

由于程序中经常有大量对文件的输入/输出操作，并且构成了程序的主要部分，例如存储数据的不单单是数据库，通过操作文件也可以保存需要的数据信息。使用这两种保存数据的方法各有千秋。使用数据库保存数据可以持久保护数据的完整性和关联性，更重要的就是安全性好，而使用文件来保存数据方便，很容易上手并且无须安装任何插件。可能之前学过其他语言的读者都知道，每种语言几乎都有自己对文件的操作方法，Python 也不例外。为了完善开发人员的需求，Python 开发人员为大家提供了使用 Python 来操作文件的方法和类。

学习目标

- 熟悉打开文件和创建文件。
- 掌握文件的增、删、改、查操作。
- 熟悉复制文件的操作。
- 熟悉修改文件名称的方法。
- 熟悉关闭文件的操作。
- 了解文件内置函数、方法和属性。
- 掌握目录的创建与删除。
- 熟悉读取目录列表。

8.1 下载页面访问量统计

在实际案例开发中，凡是提到文件操作，必然涉及文件的打开和创建。本节主要使用两个函数对文件进行打开和创建操作，其中 open()函数用于打开文件，内置函数 file()用来创建文件。通常情况下，开发人员都会使用 open()函数，而 file()函数几乎是为面向对象的程序设计的，我们通常不使用。

 **视频教学：** 光盘/videos/08/ open()函数.avi　　　　长度：6 分钟

8.1.1 基础知识——open()函数

一般情况下，使用 open()函数时只需调入文件名参数，而不添加其他任何参数，就可以获取文件内容。相反，如果要向文件中添加信息，就必须指定一个模式参数，用来声明它准备做什么，这个模式参数才是 open()的灵魂。以下代码是 open()函数的使用语法：

```
open(name[, mode[,buffering]])
```

其中，name 参数表示需要打开的文件名称，mode 是打开的模式，opne()函数的第三个参数用来控制文件的缓冲，默认值为 0，表示不缓冲，设置为 1 就会有缓冲。表 8-1 列举了 open()函数的几个模式值。

表 8-1　open()函数的模式值

参　数	描　述
r	读模式打开文件
w	读写模式打开文件
a	写入模式打开文件
b	二进制模式打开文件(可以和其他模式并用)
+	读/写模式(可以和其他模式并用)
U	支持换行符(例如：\n、\r 或\n\r 等)

也就是说，当我们使用 open()函数打开文件时，程序首先会查询 open()函数中的文件名称，接下来才是文件的模式。文件模式是相当重要的一个参数。默认情况下，该函数的默认模式参数值为 r，用来以只读方式打开文件。需要注意的是，如果读取特殊文件(例如视频或者图片文件)，那么必须使用 b 模式。

```
#-*-coding:UTF-8 -*-
#Python 模板
o=open('d:\index.txt','w');
o.write("大家好，欢迎大家光临窗内网(www.itzcn.com)！");
o.close();
```

在上述代码中，首先使用 open()函数来打开 D 盘下的 index.txt 文件，以读写模式打开，这样就可以对文件进行读写操作了。打开文件后返回一个文件对象，然后调用 write()函数写入信息，最后调用 close()关闭文件。写入函数在下一节中详细介绍。

8.1.2　实例描述

下面通过使用 Python 操作文件的有关方法创建一个文件。在开发中，我们经常会遇到将数据保存到本地文件中的操作。例如，将页面中统计页面访问量信息保存到本地文档中。在这种情况下，完全可以使用 open()函数来操作。下面的实例实现将页面访问量下载到本地。

8.1.3　实例应用

【**例 8-1**】创建第一个 doc 文档。

通过使用 Python 在 Windows 下 D 盘的 itzcn 文件夹中创建一个文件，文件名称为 statistics.doc。创建成功之后，打开文件并在文件中写入内容。

```
#-*-coding:UTF-8 -*-
#Python 模板
print "===========欢迎使用窗内网网站流量统计下载系统===========";
yn=int(raw_input("您确定要下载吗？1:表示是,2:表示否！"));
if yn==1:
    files=raw_input("请输入您需要下载的本地地址");
    o=open(files+'\\statistics.doc','w');
    print "正在下载中，请稍后......";
    o.write('【昨天】中访问量：25624，独立访客：21301，独立 IP：19813【今天】中访问量：
12453，独立访客：11523，独立 IP：11210');
    o.close();
    print "下载完成请查阅！"
else:
    print "欢迎使用，希望下次再来！";
```

以上代码首先判断当前用户是否允许下载，如果选择下载则执行 open()函数，打开文件。该函数在这里指定了两个参数，第一个是需要下载文件的名称和路径，第二个是读取模式，这里设置为 w，表示读写功能。open()函数有一种自动判断的功能，它会查找已打开的文件路径下是否存在该文件，如果不存在则创建文件，创建成功之后并打开该文件，返回文件对象，接下来调用 write()函数写入内容，关闭读写。

8.1.4　运行结果

运行代码，在 itzcn 目录下没有任何文件，先创建该文件，然后将内容写入文件中，运行结果如图 8-1 所示。

```
Python Shell
File  Edit  Shell  Debug  Options  Windows  Help
IDLE 1.2.4
>>> print "===========欢迎使用窗内网网站流量统计下载系统===========";yn=int(raw_
input("您确定要下载吗? 1:表示是,2:表示否! "));
===========欢迎使用窗内网网站流量统计下载系统===========
您确定要下载吗? 1:表示是,2:表示否! 1
>>> if yn==1:
        files=raw_input("请输入您需要下载的本地地址");
        o=open(files+'\\files.doc','w');
        print "正在下载中，请稍后......";
        o.write('【昨天】中访问量：25624，独立访客：21301，独立IP：19813【今天】中
访问量：12453，独立访客：11523，独立IP：11210');
        o.close();
        print "下载完成请查阅！"
else:
        print "欢迎使用，希望下次在来！";


请输入您需要下载的本地地址d:\itzcn
正在下载中，请稍后......
下载完成请查阅!
>>> |
Ln: 29 Col: 4
```

图 8-1　访问量统计下载

8.1.5　实例分析



源码解析

从上面的实例可以得知，通过使用 open()函数可以进行创建文件和打开文件操作，以后对文件的操作更加方便，因为打开文件后返回的信息就是一个文件对象，而且打开文件的模式也是非常有意义的参数。

8.2　创建本地记事本

上一节学习了使用 Python 创建文件，但是大家都知道仅仅创建一个空文件并没有任何意义。接下来将讲解文件的读取和写入操作，这样才能发挥文件的作用。



视频教学：光盘/videos/08/文件的读取和写入.avi　　　　长度：15 分钟

8.2.1　基础知识——文件的读取

将文本文件的内容读入，可以操作字符串变量的函数有 3 个，但它们对文件的读取方式各不相同。其中，read()函数可一次性读取数据，readline()函数按行读取，而 readlines()函数则以多行方式读取。下面针对这 3 种不同的读取方式进行详细讲解。

1．read()函数

read()函数可以一次性将文件中的所有数据读取出来，这是最简单的文件读取方式。该函数的语法如下：

```
content=read([num])
```

在语法中 read()函数只有一个参数 num，该参数主要作用就是用来控制使用该函数读取数

据时返回的字节，也就是说读取文件中的字节数。

```
#-*-coding:UTF-8 -*-
#Python 模板
o=open("d:\itzcn\statistics.txt");
content=o.read(16);
print content;
print o.tell();
content=o.read();
print content;
o.close();
```

在上述代码中，首先使用 open()函数打开文件，接下来调用 read()函数，并在该函数中传入数值 16，意思是说读取文件中前 16 个字符内容，然后显示出来。后面的 tell()方法用来查询当前文件中字符的长度，返回值是一个整数。之后使用 read()函数，但并未传入任何值，表示查询该文件的所有内容。最后关闭读取流。

2. readline()函数

readline()函数同样用来读取文件数据，但是该函数与 read()函数不同的是，它每次只读取文件中的一行数据。该函数的语法如下：

```
content=readline([num])
```

readline()函数有一个参数，该参数的作用和 read()函数的参数相同，用来返回结果集中的字符内容。

```
#-*-coding:UTF-8 -*-
#Python 模板
o=open("d:\itzcn\statistics.txt");
content=o.readline(10);
print content;
content=o.readline();
print content;
o.close();
```

文件里面保存了两行数据记录，当使用 readline()函数读取文件内容时，只会读取文件中的第一行数据，如果要读取所有内容，就需要通过循环来读取，而在 readline()函数中添加参数，则表示在当前读取的结果集中显示指定字符长度内容。

3. readlines()函数

该函数和以上介绍的两个函数都不太一样，使用该函数读取数据需要通过循环方可显示内容，主要原因在于该函数用来读取文件中的所有行内容。返回结果是一个列表集，需要进行遍历。该函数的语法如下：

```
listcontent=readlines()
```

该函数没有参数，因为返回的是一个结果集，所以不用控制显示的字符数。下面通过一个简单的例子作讲解。

```
#-*-coding:UTF-8 -*-
#Python 模板
o=open("d:\itzcn\statistics.txt");
```

```
contents=o.readlines();
for content in contents:
    print content;
o.close();
```

上述代码所读取的文件中保存了两行信息，通过使用 readlines()函数读取文件内容，返回结果是一个列表集合，因此需要使用 for 循环将列表数据循环展示。



在每次操作文件之后都要调用 close()函数，主要作用是释放资源。

8.2.2 基础知识——文件的写入

将内容通过 Python 写入已创建的文件中，这个过程叫做文件写入。在 8.1 节中讲解创建打开文件时就简单讲解过文件的写入，细心的读者可能还有印象。下面讲解在 Python 中如何使用文件写入函数 write()写入字符串以及使用函数 writelines()写入列表集到文件的有关操作及方法。

1. write()函数

该函数用来将文本字符串写入已经创建的文件中，该函数的语法如下：

```
write(content);
```

write()函数只有一个参数，代表需要写入文本的字符串。下面通过一个简单的例子来讲解如何使用该函数。其实该函数已经在前面简单提过，下面的例子大家肯定很容易明白。

```
#-*-coding:UTF-8 -*-
#Python 模板
o=open("d:\itzcn\statistics.txt","w+");
o.write("【昨天】中访问量：25624，独立访客：21301，独立 IP：19813\n【今天】中访问量：12453，独立访客：11523，独立 IP：11210");
o.close();
```

在以上代码中，首先通过 open()函数以读写方式打开文件，其中的+操作符表示在插入内容中可以包含特殊的换行符。插入文件后将会自动换行而不是显示换行符。接下来调用 write()函数，其中的参数代表需要插入文件的内容。

2. writelines()函数

writelines()函数也用于对文件进行写入操作，但是该函数用来写入一个列表内容。该函数的语法如下：

```
writelines(listcontent)
```

该函数有一个参数，该参数是一个集合类型，表明在调用时插入的数据必须是一个集合类型数据。下面通过一个简单的例子作讲解。

```
o=open("d:\itzcn\statistics.txt","w+");
listcontent=["【昨天】中访问量：25624，独立访客：21301，独立 IP：19813\n","【今天】中访问量：12453，独立访客：11523，独立 IP：11210"];
o.writelines(listcontent);
```

```
o.close();
```

以上代码首先创建了一个 listcontent 集合变量，每条数据都保存到集合中，然后调用 writelines()函数将集合传入该函数，最后关闭文件流。这时候在本地文件中将会出现集合中的所有数据信息。

8.2.3　实例描述

本实例将实现一个简单的记事本。思路非常简单，只要用户输入内容及保存地址，就可以将内容保存到本地。当需要查看内容时，只要简单地选择就可以将内容显示出来。

8.2.4　实例应用

【例 8-2】创建记事本。

本实例主要通过以读写方式来保存信息，当用户选择不同功能时，可决定该系统的操作是查询还是插入，详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
print "==========欢迎使用窗内网记事本==========";
con=True
while con:
    k=int(raw_input("请输入您的操作\n1.【我要写日记】\n2.【查看往事】\n3.【退出】\n"));
    if (k==1):
        o=open("d:\itzcn\mylog.log","a+");
        content=raw_input("请输入您需要记录的事情：\n");
        o.write(content);
        o.close();
        print "==============================";
    elif (k==2):
        print "日志内容：\n";
        o=open("d:\itzcn\mylog.log","a+");
        listcontent=o.readlines();
        for content in listcontent:
            print content;
        o.close();
        print "==============================";
    else:
        print "欢迎下次使用！";
        con=False;
```

在以上代码中，首先通过使用 while 循环让用户做完某些操作后还可以继续操作。当用户选择了【退出】选项时，则循环终止。在程序运行期间，首先获取用户输入的选项，然后做出相应的操作。当用户选择【我要写日记】选项时，使用 open()函数来创建或者打开参数内的文件。获取用户需要保存的日记内容并调用 write()函数将内容保存到文件中，这样就完成了写日记的功能。如果用户选择了【查看往事】选项，同样调用 open()函数打开文件，使用 readlines()函数获取每行数据信息，然后以 for 循环形式将文件的内容显示在控制台上，最后关闭文件。

8.2.5 运行结果

运行上述实例代码，作出相应操作，结果如图 8-2 所示。

```
Python Shell
File Edit Shell Debug Options Windows Help
>>>
==========欢迎使用窗内网记事本==========
请输入您的操作
1.【我要写日记】
2.【查看往事】
3.【退出】
1
请输入您需要记录的事情:
郑州汇智计算机技术服务有限公司成立于2002年。致力于把基于先进信息技术的最佳管理与业务实践普及到客户
的管理与业务创新活动中，全面提供企业管理ERP软件、服务与解决方案，是中原地区较早的管理软件、客户关系
管理软件、小型企业管理软件以及网站建设服务提供商。
==============================
请输入您的操作
1.【我要写日记】
2.【查看往事】
3.【退出】
2
日志内容:
郑州汇智计算机技术服务有限公司成立于2002年。致力于把基于先进信息技术的最佳管理与业务实践普及到客户
的管理与业务创新活动中，全面提供企业管理ERP软件、服务与解决方案，是中原地区较早的管理软件、客户关系
管理软件、小型企业管理软件以及网站建设服务提供商。
==============================
请输入您的操作
1.【我要写日记】
2.【查看往事】
3.【退出】
3
欢迎下次使用!
>>> |
Ln: 37 Col: 4
```

图 8-2　本地记事本

8.2.6 实例分析

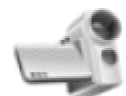


源码解析

上述实例中共用到了 4 个函数，其中 open()函数用来打开文件，write()函数用来向文件中写入内容，readlines()函数用来读取文件内容，而 close()函数用来关闭文件流。除此之外，对文件的读写操作还有其他一些函数，每个文件操作函数都有各自的优点和用途，在使用文件读取函数时要做到适可而止。

8.3 格式化本地记事本

删除文件在文件操作方面也是一个比较重要的功能，相当于当数据库内垃圾数据过多时需要删除的性质相同。这样可以大大减少硬盘空间占用，方便操作人员对内容进行管理。本节将针对文件删除操作进行详细讲解。



视频教学：光盘/videos/08/ remove()函数.avi　　长度：7 分钟

8.3.1　基础知识——remove()函数

在讲解文件删除之前，有必要认识一下 os 模块和 os.path 模块，因为在对文件进行删除时需要应用到这两个模块。Python 标准库中的 os 模块包含了普通的操作系统功能。如果你希望程序能够与平台无关，这个模块尤为重要。os 模块提供了统一的操作系统接口函数，这些接口函数通常指定了操作平台。os 模块能在不同操作系统平台下对特定函数自动切换，从而实现了跨平台操作。

表 8-2 列出了 os 模块中比较常用和重要的部分函数。

表 8-2　os模块函数

函　数	描　述
os.sep	可以取代操作系统特定的路径分割符
os.name	指示你正在使用的平台。比如对于 Windows，它是 nt，而对于 Linux/Unix 用户，它是 posix
os.getcwd()	得到当前工作目录，即当前 Python 脚本的目录路径
os.getenv()	读取环境变量
os.putenv()	设置环境变量
os.listdir()	返回指定目录下的所有文件和目录名
os.remove()	删除一个文件
os.system()	运行 shell 命令
os.path.split()	返回一个路径的目录名和文件名
os.path.isfile()	检验路径是不是文件
os.path.isdir()	检验路径是不是目录
os.path.existe()	检验给出的路径是否真正存在
os.listdir(dirname)	列出 dirname 下的目录和文件
os.getcwd()	获得当前工作目录
os.chdir(dirname)	改变工作目录到 dirname
os.path.isdir(name)	判断 name 是不是一个目录，name 不是目录则返回 false
os.path.isfile(name)	判断 name 是不是一个文件，不存在 name 则返回 false
os.path.exists(name)	判断是否存在文件或目录 name
os.path.getsize(name)	获得文件大小，如果 name 是目录则返回 0L
os.path.abspath(name)	获得绝对路径
os.path.split(name)	分割文件名与目录
os.path.join(path,name)	连接目录与文件名或目录
os.path.basename(path)	返回文件名
os.path.dirname(path)	返回文件路径

表 8-2 中的函数都是 os 模块下的常用函数，在对文件进行删除操作时需要应用到该模块下

的 remove()函数。该函数的语法如下：

```
import os;
os.remove(file)
```

可以得知，在调用 os 模块中的函数前需要将该模块引入工程。调用 remove()函数时有一个参数，该参数表示需要删除的文件地址。代码实例如下：

```
import os;
o=open("d:\itzcn\statistics.txt");
content=o.read(16);
print content;
o.close();
if os.path.exists("d:\itzcn\statistics.txt"):
    os.remove("d:\itzcn\statistics.txt");
    print "删除成功！";
```

在以上代码中，首先使用 open()函数打开文件，并调用 read()函数来读取文件中的数据，然后使用 os.path.exists()函数来判断当前目录下的文件是否存在，如果存在则调用 os.remove()函数进行删除操作。exists()和 remove()函数中的参数均为文件的物理路径。

8.3.2 实例描述

该实例继续完善本地记事本，在上节内容中讲解了记事本的创建以及记事本的内容展示功能，但仅仅具备这些功能肯定不是一个成熟的记事本。下面就对记事本进行完善，在这里将对记事本添加格式化功能，方便用户对没用的日记进行删除处理，所谓格式化记事本就是将记事本删除，然后重新创建。

8.3.3 实例应用

【例 8-3】格式化记事本。

当用户选择【格式化记事本】选项时，提示用户再次确认是否要格式化，用户输入 1 表示格式化，程序判断记事本是否存在，如果存在则执行删除记事本的操作，然后创建空记事本，这样就实现了格式化记事本的功能。

```
#代码省略
    elif (k==3):
        gsh=int(raw_input("您确定要格式化记事本吗?格式化后数据将会全部消失\n 确定输入
1，取消输入 2：\n"));
        if (gsh==1):
            print "记事本正在格式化中......";
            if os.path.exists("d:\itzcn\mylog.log"):
                os.remove("d:\itzcn\mylog.log");
                print "记事本格式化成功！";
                open("d:\itzcn\mylog.log","a+");
            else:
                print "记事本不存在！";
#代码省略
```

在以上代码中，主要使用 os.path.exists()函数来判断文件是否存在，然后使用 os.remove()函数删除文件，删除后再使用 open()函数创建新的记事本。

8.3.4　运行结果

运行实例代码，作出相应的操作，结果如图 8-3 所示。

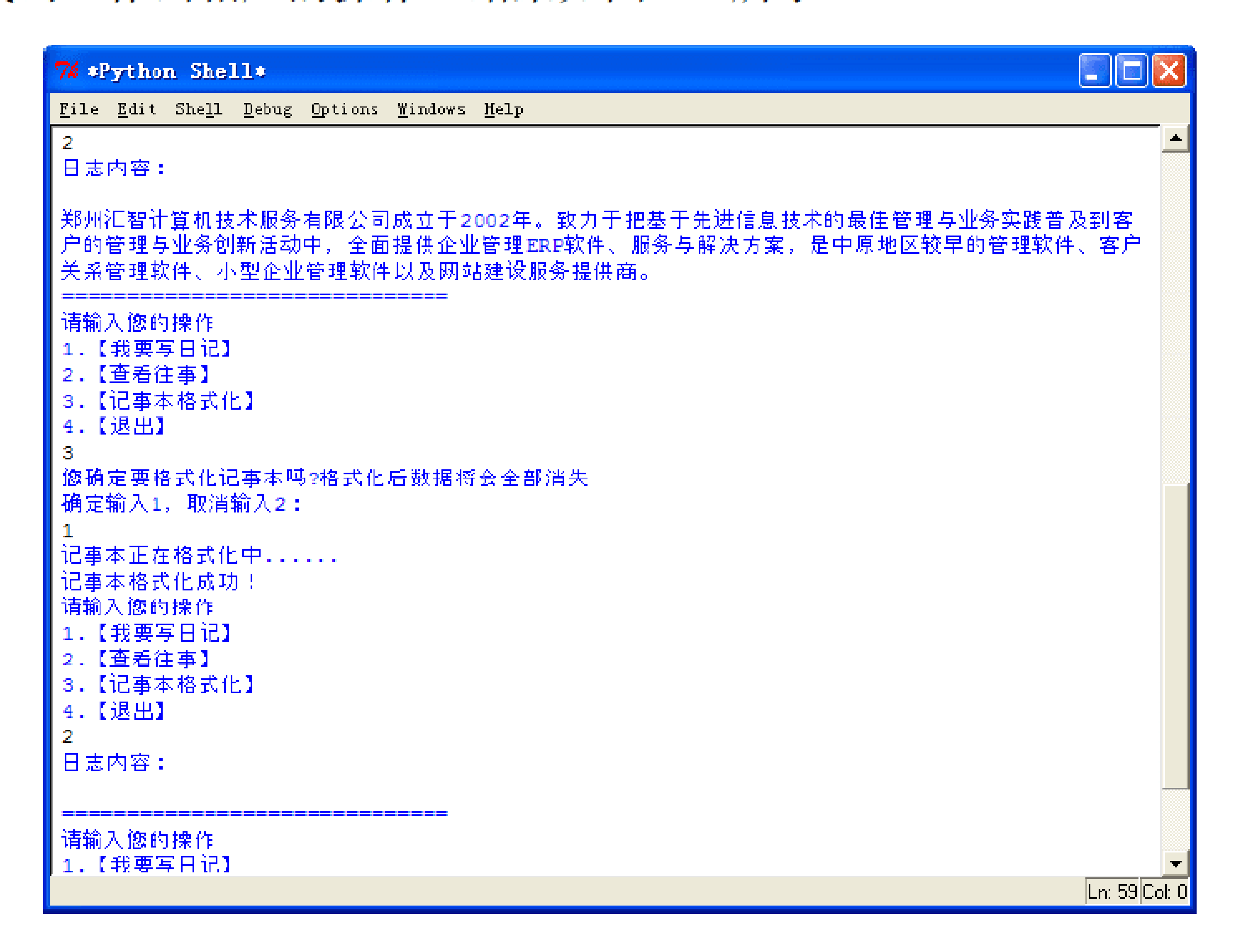


图 8-3　记事本格式化

8.3.5　实例分析

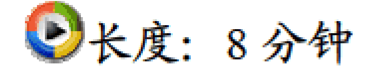


源码解析

在实例中主要使用 os 模块下的 path.exists()函数来判断文件是否存在，使用 path.exists()来对文件执行删除操作。在 os 模块下还有其他常用的函数，例如使用 path.basename()函数返回文件名，或者使用 os.path.basename()函数返回文件路径等。位于 os 模块下的函数对文件的操作还有很多强大的功能。

8.4　备份与恢复本地记事本

文件的复制功能也是比较重要的功能之一，它能减轻用户操作文件的诸多负担，在文件管理方面起着不可或缺的重要作用。有了文件复制功能，可以将文件复制一份作为备份文件，以防文件丢失。

**视频教学：**光盘/videos/08/ copyfile()和 move()函数.avi　

8.4.1 基础知识——copyfile()和move()函数

在讲解对文件复制和移动之前，还需要了解一下 shutil 模块。shutil 模块是一种高层次的文件操作工具，类似于高级 API，其强大之处在于对文件的复制、备份操作非常重要。表 8-3 列举了该模块下的一些常用函数。

表 8-3 shutil模块函数

函 数	描 述
copyfile(src,dst)	从源 src 复制到 dst 中去，前提是目标地址具备可写权限。抛出的异常为 IOException。如果当前 dst 已存在就会被覆盖
copymode(src,dst)	只复制其权限，其他东西不会被复制
copystat(src,dst)	复制权限、最后访问时间、最后修改时间
copy(src,dst)	复制一个文件到另一个文件或另一个目录
copy2(src,dst)	在复制的基础上再复制文件最后访问时间与修改时间
copy2(src,dst)	如果两个位置的文件系统是一样的，相当于执行了 rename 操作，只是改名；如果是在不相同的文件系统下，就做 move 操作
copytree(olddir, newdir,True/Flase)	把 olddir 复制一份 newdir，如果第 3 个参数是 True，则复制目录时将保持文件夹下的符号连接；如果第 3 个参数是 False，则将在复制的目录下生成物理副本来替代符号连接

在不了解 shutil 模块具有对文件复制的功能之前，可以通过使用 read()和 write()函数来制作一个模拟复制功能。学习该模块之后，可以通过使用 shutil 模块中的 copyfile()函数来对文件进行复制操作，函数语法如下：

```
copyfile(src,dst)
```

该函数有两个参数，其中 src 表示需要复制文件的地址，参数 dst 表示文件复制的目的地址。另外，在 shutil 模块下还可以使用 move()函数来实现文件移动操作，该函数的参数和 copyfile()函数的参数类似，详细代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
import shutil;
shutil.copyfile("d:\itzcn\mylog.log","d:\\itzcn\\bf_mylog.log");
shutil.move("d:\\itzcn\\bf_mylog.log","d:\\itzcn\\bf\\bf_mylog.log");
```

在以上代码中，使用 import 关键字将 shutil 模块引入工程，然后调用 copyfile()函数复制文件并重新命名为 bf_mylog.log，复制成功之后再使用 move()函数指定到 bf 目录。

8.4.2 实例描述

该实例继续完善本地记事本。任何项目都是在不断地完善之后才会成熟。下面将在本地记事本中添加一个日记备份功能，这样可以避免记事本在出现错误时无法恢复数据的缺陷，有了这个功能你就可以放心地使用这款记事本了。

8.4.3　实例应用

【例 8-4】备份及恢复本地记事本。

该功能主要通过使用 shutil 模块中的文件复制和文件移动功能来实现。既然有了数据备份肯定会有数据恢复功能，所以使用该方法也可以实现数据恢复操作。

```
#代码省略
elif (k==4):
    bf=int(raw_input("您确定要备份记事本吗?\n 确定输入 1，取消输入 2：\n"));
    if (bf==1):
        print "记事本正在备份中......";
        shutil.copyfile("d:\itzcn\mylog.log","d:\\itzcn\\bf_mylog.log");
        shutil.move("d:\\itzcn\\bf_mylog.log","d:\\itzcn\\bf\\bf_mylog.log");
elif (k==5):
    hf=int(raw_input("您确定要恢复记事本吗?\n 确定输入 1，取消输入 2：\n"));
    if (hf==1):
        print "记事本正在恢复中......";
        shutil.copyfile("d:\\itzcn\\bf\\bf_mylog.log","d:\\itzcn\\mylog.log");
#代码省略
```

程序首先判断用户的选择，当用户选择【备份记事本】选项时，执行数据备份代码。在以上代码中，使用 copyfile()函数来对文件进行复制，使用该函数的前提是，事先将该模块引入工程。复制成功之后将该文件移动至安全位置保存，这里使用 move()函数将文件移动到 bf 目录下。如果用户选择了【记事本恢复】选项，则使用 copyfile()函数将备份文件复制到日记文件目录下并进行覆盖。

8.4.4　运行结果

运行代码然后对数据进行备份和恢复操作，结果如图 8-4 所示。

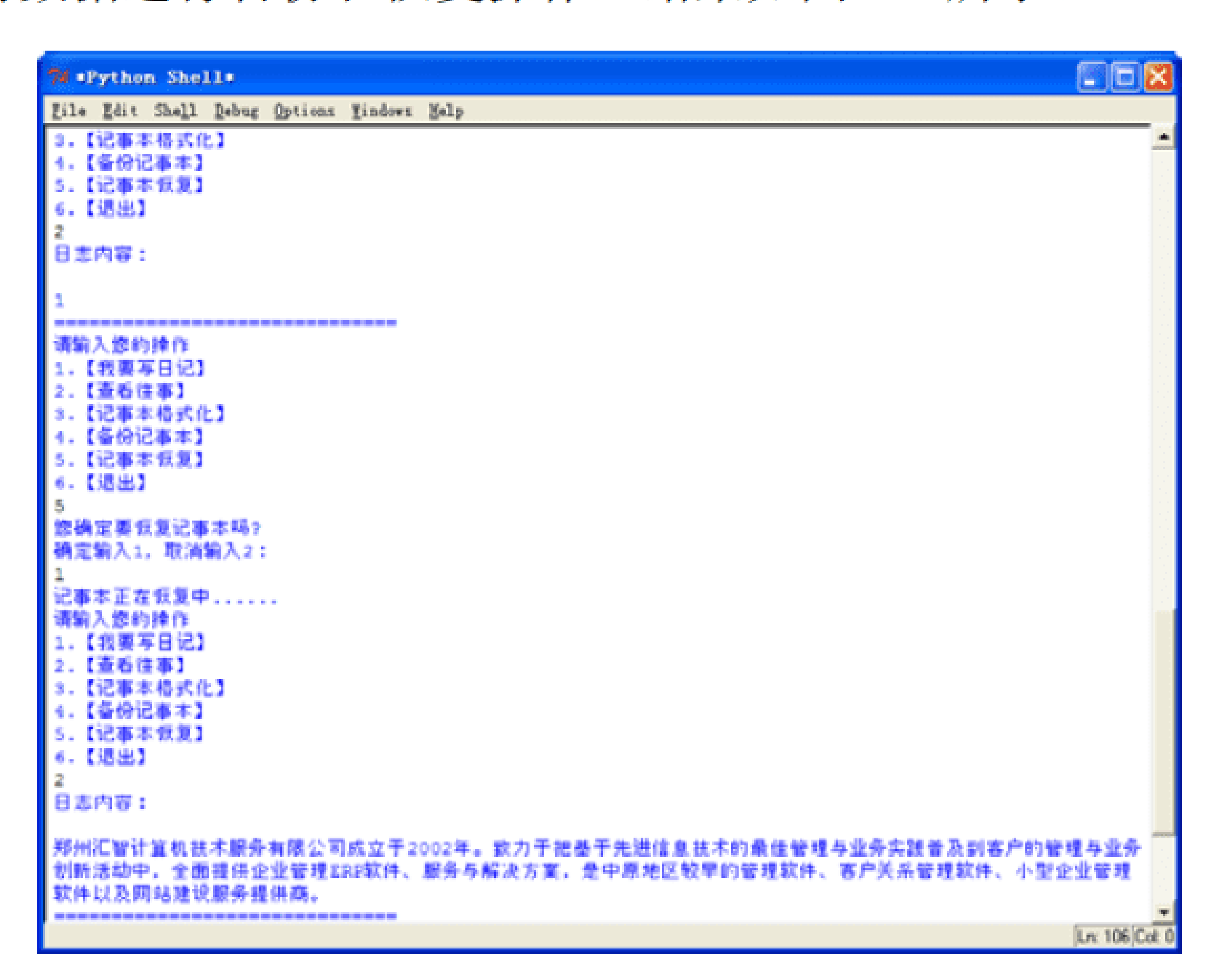


图 8-4　日记备份与恢复

8.4.5 实例分析



源码解析

该实例通过使用 shutil 模块下的部分方法来完成数据备份及恢复功能。在该模块下有一些对文件进行复制、删除、重命名和移动等操作函数，必要时还可以借助该模块下的其他函数来实现程序的需求。

8.5 日记内容过滤器

对文件内容的查找与替换的好处在于，可以减少用户对文件内容的操作频率。通过使用文件内容查找功能，可以让用户非常方便地找到自己想要的内容。如果使用传统的查找方式，当内容非常少的时候尚可行，但当内容过多甚至超过数 GB 时，可能就等同于大海捞针了。使用文件内容查找功能可以减少不必要时间的浪费。同理，文件内容替换也是如此，如果手动一个个地替换，可能会浪费大量的时间，可是通过使用程序来自动替换却能节省很多时间。



视频教学：光盘/videos/08/日记内容过滤器.avi　　　　长度：4 分钟

8.5.1 实例描述

在前面章节中我们已经讲解了文件内容查找与替换的函数，不知道大家是否还有印象？没错，在讲解字符串与正则表达式的章节中我们已经介绍了如何使用该函数，这里同样可以将该函数应用到文件内容操作上。

下面继续完善本地记事本。在本地记事本中添加一个内置的日记内容过滤器功能，也就是说当用户输入日记内容进行过滤时，例如当用户输入“窗内网”关键字时，可以将内容替换为“窗内网(http://www.itzcn.com)”的效果。

8.5.2 实例应用

【例 8-5】日记内容过滤器。

该实例是在本地记事本系统中日记添加功能的基础上在完善的一个实例，通过获取用户输入信息对内容的部分字段进行替换，详细代码如下所示。

```
#代码省略
if (k==1):
    o=open("d:\itzcn\mylog.log","r+");
    content=raw_input("请输入您需要记录的事情：\n");
    count=0;
    for s in o.readlines():
```

```
        li=re.findall("窗内网",s);
        if len(li)>0:
            count=count+li.count("窗内网");
    th=int(raw_input("查找到"+str(count)+"个可能替换的内容是否继续？\n 确定输入 1，
取消输入 2：\n"));
    if th==1:
        content=content.replace("窗内网","窗内网(http://www.itzcn.com)");
    o.write(content);
    o.close();
    print "===============================";
#代码省略
```

在以上代码中，首先获取用户输入的日记内容，并通过循环判断，使用 findall()函数来查询当前文本中存在多少个需要替换的内容，然后提示用户是否继续执行。当用户选择继续执行时则执行 replace()函数进行数据替换操作，即将“窗内网”替换为“窗内网(http://www.itzcn.com)”之后并将内容写入文件。

8.5.3　运行结果

运行实例，然后编写日记，输入“窗内网提供”字段，结果如图 8-5 所示。



图 8-5　日志信息替换

8.5.4　实例分析

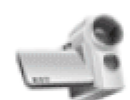


源码解析

readlines()和 replace()函数在之前章节中已经讲到了，这里就不过多解释它们的用法和作用了。本章主要告诉大家的就是用于文本操作的函数同样可以用于文件内容操作。这样的函数在 Python 程序中是通用的。

8.6 记事本的分类

所谓目录就是 Windows 系统中的文件夹，有了文件夹，方便我们对各种文件进行分类整理，主要目的是更好地保存文件，使它整齐规范。前面已经讲解了文件的一些操作，有了文件何不学习一下文件夹呢？下面将介绍使用 Python 函数创建与删除目录。

 视频教学：光盘/videos/08/记事本的分类.avi　　长度：18 分钟

8.6.1 基础知识——mkdir()函数

该函数用来创建一个文件目录。通过使用该函数，可以在指定的路径中创建自定义目录，该函数的语法如下：

```
import os;
os.mkdir(dir)
```

mkdir()函数位于 os 模块下，它只有一个参数，表示创建目录的路径以及目录名称，实例代码如下：

```
import os;
os.mkdir("d:\\itzcn\\2011-3-15");
open("d:\\itzcn\\2011-3-15\\statistics.txt","a+");
```

以上实例代码通过使用 mkdir()函数在 itzcn 目录下创建一个 2011-3-15 子目录，然后使用 open()函数在该目录下创建一个文本文件。

8.6.2 基础知识——makedirs()函数

该函数同样用来创建一个目录，它和 mkdir()函数的作用相同。唯一不同的是，makedirs()函数可以创建多级目录，而 mkdir()函数则不能。该函数的语法如下：

```
os.makedirs(dir)
```

和 mkdir()函数的语法类似，只是用该函数创建的路径可以是多级目录，如果强制使用 mkdir()函数创建两个或者两个以上的目录则会抛出异常。下面实例通过使用 makedirs()函数创建多级目录，代码如下：

```
import os;
os.makedirs("d:\\itzcn\\2011-3-15\\bf2011-3-15");
open("d:\\itzcn\\2011-3-15\\statistics.txt","a+");
open("d:\\itzcn\\2011-3-15\\bf2011-3-15\\bfstatistics.txt","a+");
```

实例中在 itzcn 文件夹中没有任何文件，通过使用 makedirs()函数，在该目录下创建一个名为 2011-3-15 的文件夹，之后在 2011-3-15 文件夹下再创建了 bf2011-3-15 子文件夹，然后在每个文件夹中都创建一个文本文件。

8.6.3　基础知识——rmdir()函数

rmdir()函数用来删除空目录，如果删除的目录中存在子文件夹或者文件时则会抛出异常，该函数的语法如下：

```
os.rmdir(dir);
```

该函数只有一个参数，表示需要删除的目录。当用户将该参数设置为文件路径时将会抛出异常，只有指定为一个目录路径时才会执行删除操作，代码如下：

```
import os;
os.rmdir("d:\\itzcn\\2011-3-15\\bf2011-3-15");
```

在上述代码中，通过使用 os 模块下的 rmdir()函数将 2011-3-15 目录下的 bf 2011-3-15 目录删除，如果使用该函数删除 2011-3-15 目录则会抛出异常，因为在该目录下还有一个目录。

8.6.4　基础知识——rmtree()函数

可能有人会有这样的疑问：如果要删除一个非空目录，该怎么办呢？不要急，下面将介绍使用 rmtree()函数来删除非空目录。该函数的语法如下：

```
shutil.rmtree(dir)
```

rmtree()函数位于 shutil 模块下，而不在 os 模块下，因此大家在使用该函数时一定要注意不要导错了模块。下面使用该函数删除非空目录。

```
import shutil;
import os;
os.mkdir("d:\\itzcn\\2011-3-15");
open("d:\\itzcn\\2011-3-15\\statistics.txt","a+");
shutil.rmtree("d:\\itzcn\\2011-3-15");
```

在上述代码中，首先使用 os 模块下的 mkdir()函数创建一个名为 2011-3-15 目录，然后使用 open()函数在 2011-3-15 目录中创建一个文本文件。创建成功之后，使用 shutil 模块下的 rmtree()函数对该文件进行删除。在没有执行删除之前，2011-3-15 目录是非空的，所以使用 rmtree()函数可以将非空目录删除。

8.6.5　实例描述

本实例将给本地记事本添加分类功能。可对【我要写日记】、【查看往事】、【查看往事】和【备份记事本】选项进行修改，主要是将每天记录的日记保存在不同的目录中，同时在执行查看和备份时都需要输入记事本的分类名称。

8.6.6 实例应用

【例 8-6】记事本分类。

用户填写日记，程序将内容保存到不同的目录下，代码如下：

```
#代码省略
if (k==1):
        times="d:\\itzcn\\"+time.strftime("%Y-%m-%d",time.localtime());
        if os.path.exists(times)==False:
            os.makedirs(times);
        files=times+"\mylog.log";
        o=open(files,"a+");
        content=raw_input("请输入您需要记录的事情：\n");
        count=0;
        for s in o.readlines():
            li=re.findall("窗内网",s);
            if len(li)>0:
                count=count+li.count("窗内网");
        th=int(raw_input("查找到"+str(count)+"个可能替换的内容是否继续？\n确定输入
1，取消输入 2：\n"));
        if th==1:
            content=content.replace("窗内网","窗内网(http://www.itzcn.com)");
        o.write(content);
        o.close();
        print "===============================";
#代码省略
```

在上述代码中，首先获取当前系统时间，作用就是将文件目录名称以时间格式保存，然后使用 os 模块下的 path.exists()函数判断该目录或者文件是否存在，如果不存在则使用 os 模块下的 makedirs()函数创建该目录同时创建日志文件。相反，如果存在此目录则打开该目录下的日志文件，并将内容保存到该文件中。日志信息保存成功之后可以查看日志，详细代码如下：

```
#代码省略
elif (k==2):
        fls=raw_input("请输入记事本分类名称：\n")
        if os.path.exists("d:\itzcn\\"+fls+"\mylog.log"):
            print "日志内容：\n";
            o=open("d:\itzcn\\"+fls+"\mylog.log","a+");
            listcontent=o.readlines();
            for content in listcontent:
                 print content;
            o.close();
        else:
            print "分类名称不存在！";
        print "===============================";
#代码省略
```

以上代码仅仅查看日志的功能，首先接受用户输入的日志类型名称，判断该目录是否存在，如果存在则打开该目录下的日志文件并且读取内容。

下面为记事本添加日志格式化功能，同样需要获取用户输入的日志分类名称，代码如下：

```
#代码省略
```

```
elif (k==3):
        gsh=int(raw_input("您确定要格式化记事本吗?格式化后数据将会全部消失\n 确定输入
1，取消输入 2：\n"));
        if (gsh==1):
            fls=raw_input("请输入格式化记事本分类名称：\n")
            print "记事本正在格式化中......";
            if os.path.exists("d:\itzcn\\"+fls+"\mylog.log"):
                os.remove("d:\itzcn\\"+fls+"\mylog.log");
                print "记事本格式化成功！";
                open("d:\itzcn\\"+fls+"\mylog.log","a+");
            else:
                print "记事本不存在！";
                print "===============================";
#代码省略
```

该功能的实现思路和上段代码的实现思路相同，即首先获取用户输入记事本分类名称，然后使用 remove()函数将分类名称下的日志移除，再创建一个新文件。

```
elif (k==4):
        bf=int(raw_input("您确定要备份记事本吗?\n 确定输入 1，取消输入 2：\n"));
        if (bf==1):
            fls=raw_input("请输入备份记事本分类名称：\n")
            print "记事本正在备份中......";
            if os.path.exists("d:\itzcn\\"+fls+"\mylog.log"):
                os.makedirs("d:\\itzcn\\"+fls+"\\bf")
                shutil.copyfile("d:\itzcn\\"+fls+"\\mylog.log",
"d:\\itzcn\\"+fls+"\\bf_mylog.log");
                shutil.move("d:\\itzcn\\"+fls+"\\bf_mylog.log",
"d:\\itzcn\\"+fls+"\\bf\\bf_mylog.log");
                print "备份成功";
            else:
                print "备份记事本分类不存在！";
```

这段代码主要实现日志文件备份的功能。在备份记事本日记前，需要使用 makedirs()函数在该目录下创建一个名为 bf 的目录，然后将备份日记文件保存在该目录下的 bf 子目录中。

8.6.7　运行结果

运行以上整理好的程序代码，然后输入相应的指令对日记进行查看、备份和恢复等操作，运行结果如图 8-6 所示。

运行结果显示的是通过使用程序对日志文件进行储存和备份后的文件存放目录结构，如图 8-7 所示。

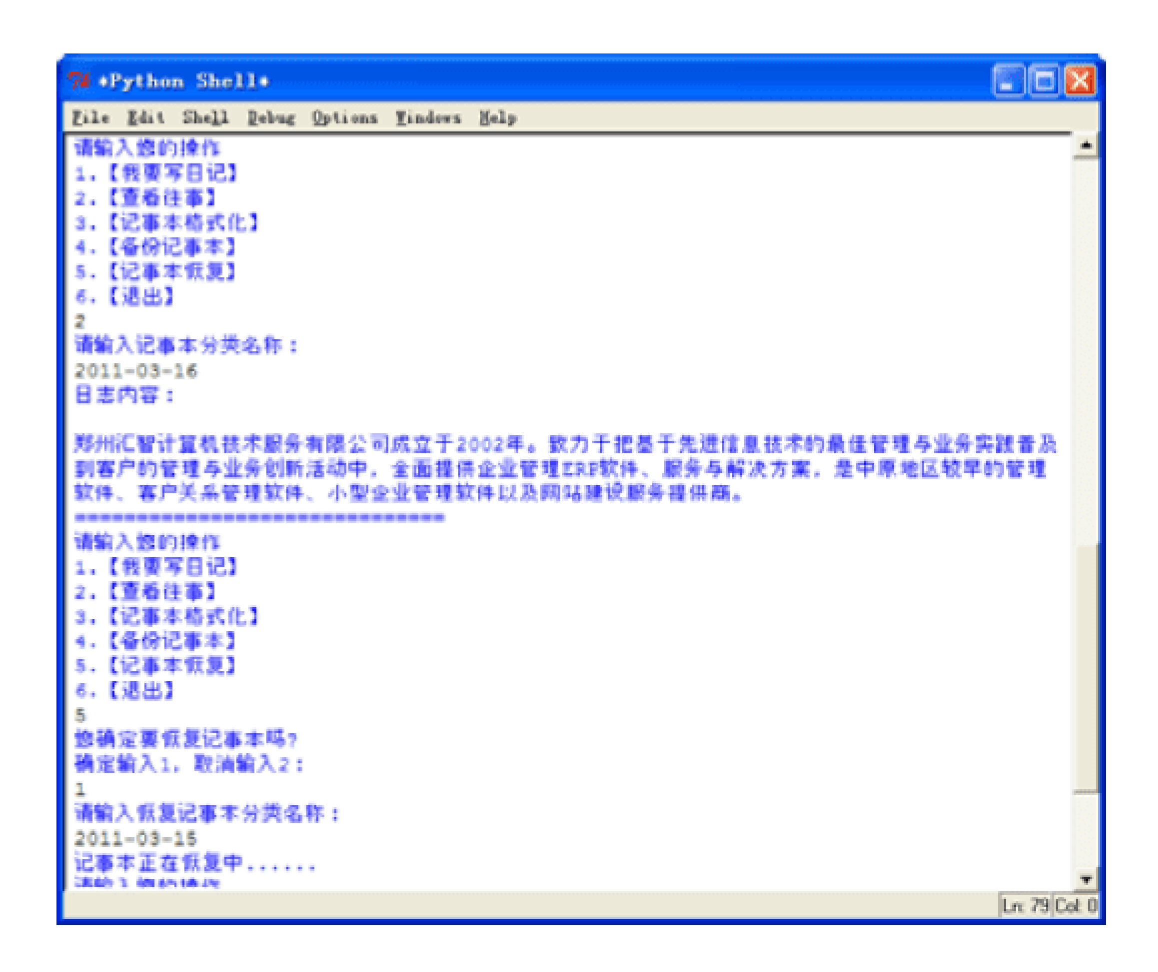


图 8-6 记事本分类

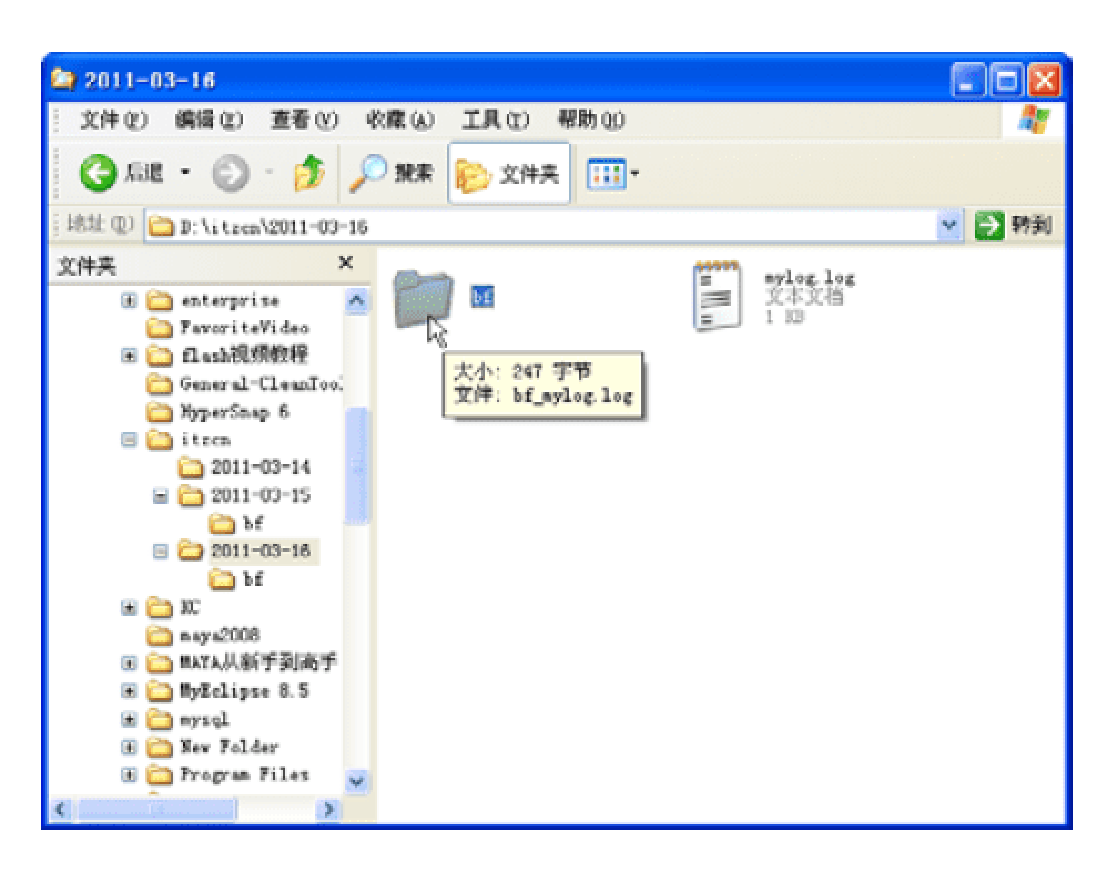


图 8-7 日志文件目录

8.6.8 实例分析



源码解析

本实例主要运用 makedirs()函数来创建不同的文件目录，从而对日记文件进行合理储存。该函数主要作用是创建多级目录，如果在应用中创建的目录是单个，那么可以使用 mkdir()函数。既然有创建肯定也有删除，但在本实例中没有用到删除函数，希望大家通过使用 rmdir()或者 rmtree()函数来为记事本制作一个删除目录的功能。

8.7 记事本文件列表

如果你的程序要对文件和目录进行操作，那么读取目录列表这个功能必定是该程序必不可少的功能之一。为了更加方便清晰地让用户通过前台得知目录的创建结果以及文件列表，减少

用户登录服务器后台来查看文件列表的次数，本节将介绍如何使用 Python 中的函数来读取目录列表。

 **视频教学：**光盘/videos/08/os.walk()和 os.path.walk()函数.avi　　**长度：**8 分钟

8.7.1　基础知识——os.walk()和os.path.walk()函数

首先讲解的第一个函数为 os 模块下的 walk()函数，该函数的作用是读取指定目录下的文件并返回，该函数的语法如下：

```
os.walk(path,fun,par)
```

既然该函数位于 os 模块下，那么在使用该函数之前需要将该函数所属模块引入当前工程。可以看到，该函数有 3 个参数，其中参数 path 表示需要遍历的目录树的路径；参数 fun 表示回调函数，对遍历路径进行处理称作回调函数，作为某个函数的参数，当某个事件被触发时，程序将调用定义好的回调函数来处理某个任务；参数 par 因第二个参数 fun 而存在，主要用来传递回调函数中的参数值。

```
#-*-coding:UTF-8 -*-
#Python 模板
import os;
def ListDir(path,fun,par):
  for filespath in par:
    print os.path.join(fun,filespath)
if __name__=="__main__":
     path="d:\itzcn";
     os.walk(path,ListDir,())
```

从上述代码可以得知，首先通过使用 import 关键字将 os 模块引入当前工程，然后使用 def 自定义一个名为 ListDir()函数，在该函数中使用 for 循环接受参数信息，使用 join()函数连接目录与文件名或目录并打印出来，最后使用 os.walk()函数并且传入相应的参数值，其参数将调用上述代码中自定义函数 ListDir()。运行代码，将把 d:\itzcn 目录下的文件进行遍历显示。

或许有人会问：使用 os.walk()函数只能遍历当前目录下的文件路径，那么如果要求遍历该目录下的子目录和文件，并且要求以树结构形式将该目录下的所有文件和目录全部遍历出来，如何实现？不要急，下面将介绍 os.path.walk()函数。

os.path.walk()函数也可以用来遍历、读取目录。它和 os.walk()函数的区别在于，os.path.walk()函数产生的文件名列表不同于 os.walk()函数产生的目录树下的目录和文件，即 os.walk()函数只产生文件路径。该函数的语法如下：

```
walk(path,topdown=True,onerror=None)
```

该语法和 os.walk()函数的语法非常相似，应用方法也相同，唯一不同之处在于参数的使用。其中参数 path 表示需要遍历的目录树路径。参数 topdown 的默认值是 True，表示首先返回目录树下的文件，然后遍历目录树的子目录。当 topdown 的值为 False 时，表示先遍历目录树的子目录，然后返回子目录下的文件，最后返回根目录下的文件。第三个参数 onerror 的默认值是 None，表示忽略文件遍历时所产生的错误。如果不为空，则提供一个自定义函数来提示错误信息，然后继续遍历或抛出异常，中止遍历。

8.7.2 实例描述

继续使用本地记事本，为该记事本添加一个日志文件列表功能。该功能的主要作用就是将用户编写的日志文件，通过使用 Python 中的方法，将路径和文件信息遍历出来，方便用户管理和查看。

8.7.3 实例应用

【例 8-7】记事本分类列表。

该实例主要用来当用户选择此项操作时，遍历日志系统的文件以及目录路径信息。详细代码如下：

```
    elif (k==6):
        print "日记分类信息：";
        def ListDir (path,fun,par):
            for filespath in par:
                print os.path.join(fun,filespath).strip("d:\itzcn");
        if __name__=="__main__":
            os.path.walk("d:\itzcn",ListDir,());
```

在上述代码中，使用 def 自定义名为 ListDir 函数，在该函数中使用循环遍历文件目录信息，然后使用 strip()函数截取所读取的路径，最后使用 os.path.walk()函数遍历列表信息。

8.7.4 运行结果

运行代码，然后选择该项操作，将会显示记事本文件列表信息，结果如图 8-8 所示。

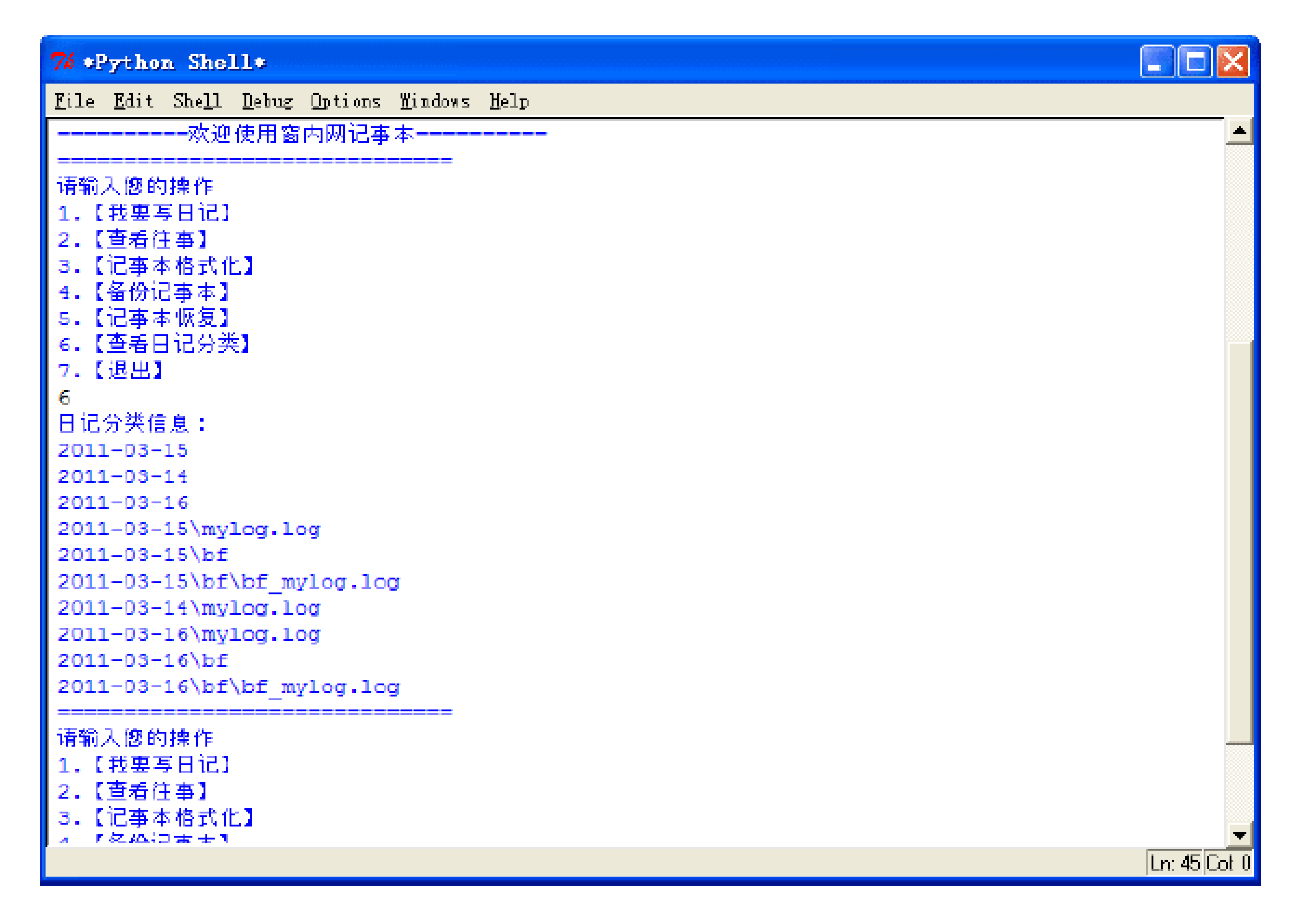


图 8-8 记事本文件列表

8.7.5　实例分析



源码解析

在本实例中使用了 os.path.walk()函数来读取目录信息的主要原因是因为该函数可以树结构来读取目录下的子目录和文件地址信息，而 os.walk()函数只能读取当前目录下的文件路径信息，但无法获取子级目录下的目录和文件信息。

8.8　常见问题解答

8.8.1　使用os模块函数出错



在使用 os 模块下的函数时出现异常信息，如何解决？
网络课堂：http://bbs.itzcn.com/thread-15264-1-1.html

今天刚学习到如何使用 os 模块下的一个名为 remove()函数来删除文件，可是运行时抛出一个莫名其妙的异常，如图 8-9 所示。

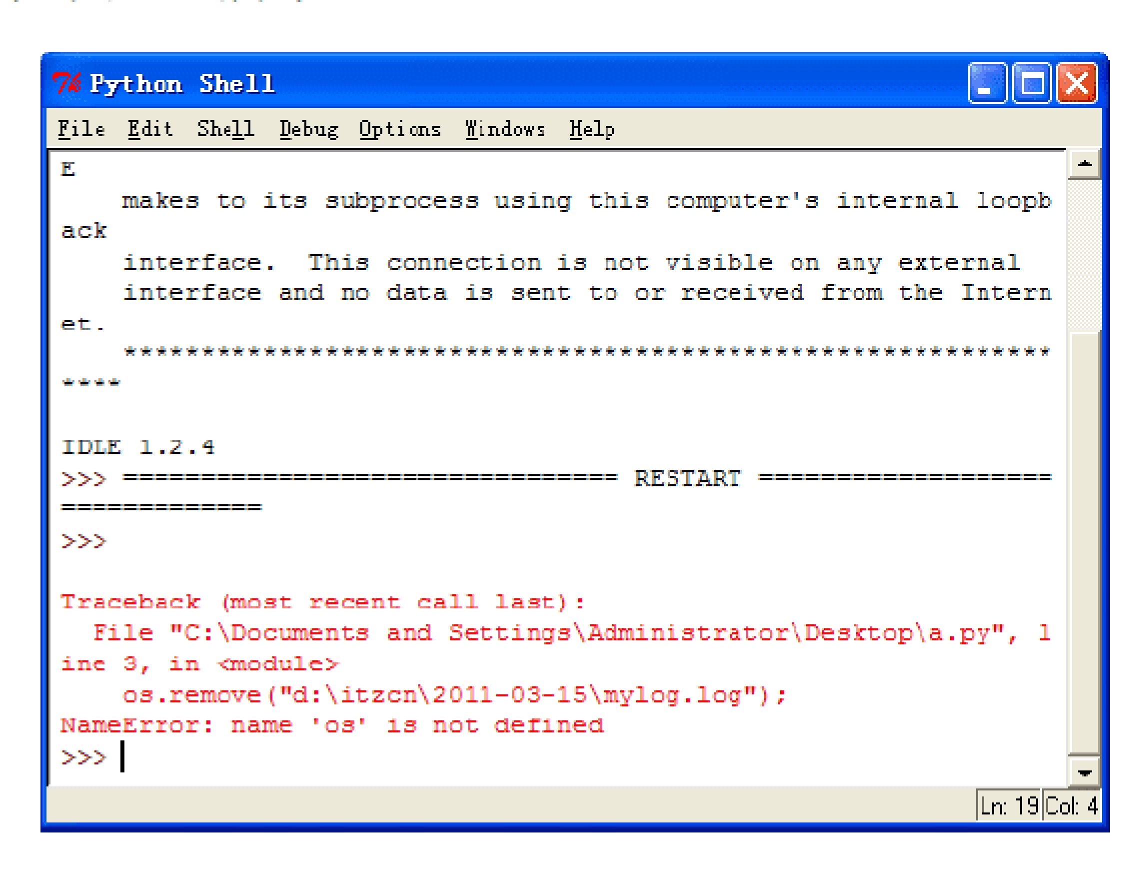


图 8-9　os模块异常

希望各位大侠帮我解决一下，我是用原版代码写的，同样出现这样的错误，代码如下：

```
#-*-coding:UTF-8 -*-
#Python 模板
os.remove("d:\itzcn\2011-03-15\mylog.log");
```

【解决办法】不用看代码就知道你是在哪里出错了。很明显找不到 os，造成这样的异常主要原因还是没有引入 os 模块库，只需在你的代码前端添加导入代码即可。

```
import os;
```

8.8.2 使用write()函数时出错



使用 write()函数时出现异常！
网络课堂：http://bbs.itzcn.com/thread-15265-1-1.html

为什么我在使用 write()函数对一个文件进行编写内容时出现异常，我确定我的语法没有错误，但是找不到错误的原因，希望大虾帮我解决一下。程序代码如下：

```
o=open("d:\\itzcn\\2011-03-15\\mylog.log","r");
o.write("欢迎来到窗内网！");
o.close();
```

语法绝对正确，而且该路径下的文件也存在，就是在写内容时出错。错误提示信息如图 8-10 所示。

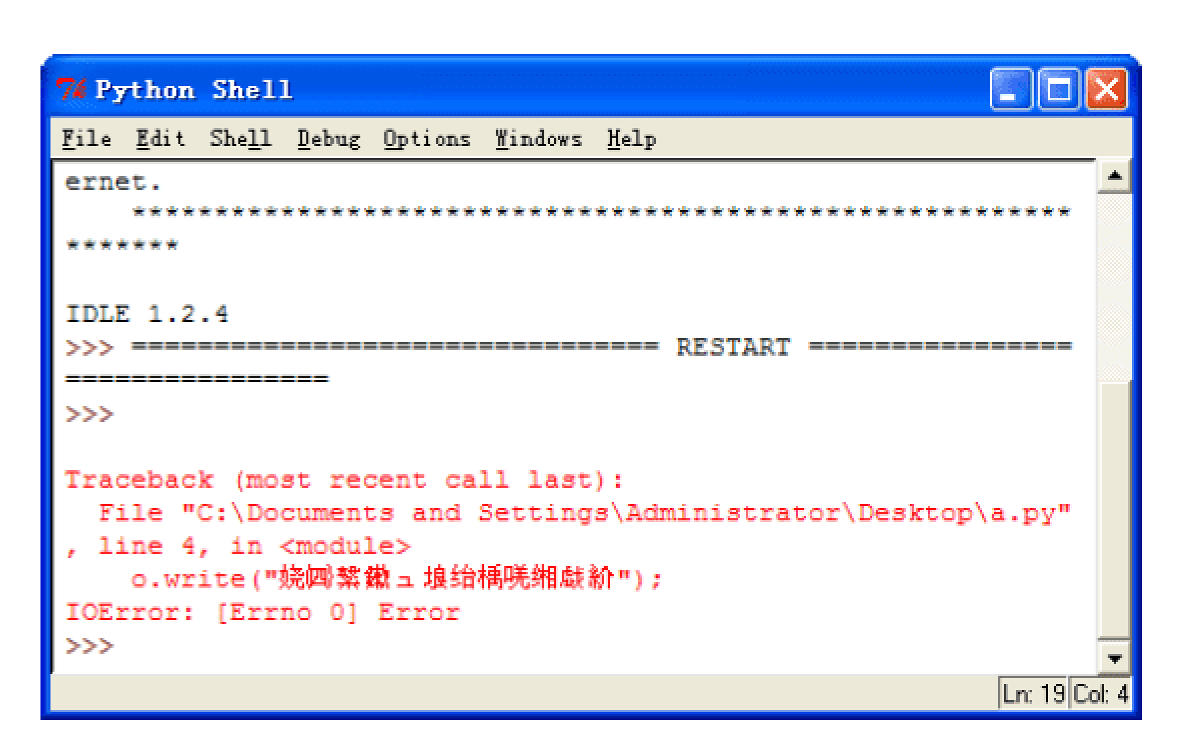


图 8-10 write()函数错误提示

【解决办法】老兄你犯了一个比较致命的错误：你的路径没有错误，语法也没有错误，唯一错误的地方就是在使用 open()函数时，第二个参数的配置有误。你是想在一个文件中填写内容，而你的代码中却使用了 r，表示读取模式。如果想对文件进行编写，必须开通读写模式。正确代码如下：

```
o=open("d:\\itzcn\\2011-03-15\\mylog.log","r+");
o.write("欢迎来到窗内网！");
o.close();
```

只需在 r 后面添加+号即可，它表示读写模式(还可以和其他模式并用)。

8.9 习　　题

一、填空题

(1) 在文件读取函数中，__________函数用来一次性读取文件中的所有数据信息。

(2) 使用 writelines()函数写入数据时，数据类型是__________。

(3) copyfile()函数用来复制一个文件，那么该函数属于__________模块。

(4) 在 os 模块下，__________函数用来删除一个文件。

(5) 在某个目录下，如果希望创建单个文件目录，该选择__________函数。

二、选择题

(1) open()函数的第二个参数表示读写模式的值，正确的为__________。

A. r　　B. w　　C. a　　D. b

(2) 下列函数中__________函数用来删除非空文件目录。

A. mkdir()函数　　B. rmdir()函数

C. remove()函数　　D. rmtree()函数

(3) 下属代码中写法不正确的是__________。

A. `o=open("d:\\itzcn\\2011-03-15\\mylog.log","r");`

B. `o=open("d:\\itzcn\\2011-03-15\\mylog.log","w");`

C. `o=open("d:\\itzcn\\2011-03-15\\mylog.log","r+");`

D. `o=open("d:\\itzcn\\2011-03-15\\mylog.log","+a");`

三、上机练习

上机练习： 记事本日志分类删除功能。

该实例非常简单，借助前面章节中讲解的内容来完成本地记事本的分类删除功能，要求使用 rmtree()或者 rmdir()函数进行文件目录操作。但要注意如果使用 rmdir()函数删除目录时一定要确保该目录为空。运行结果如图 8-11 所示。

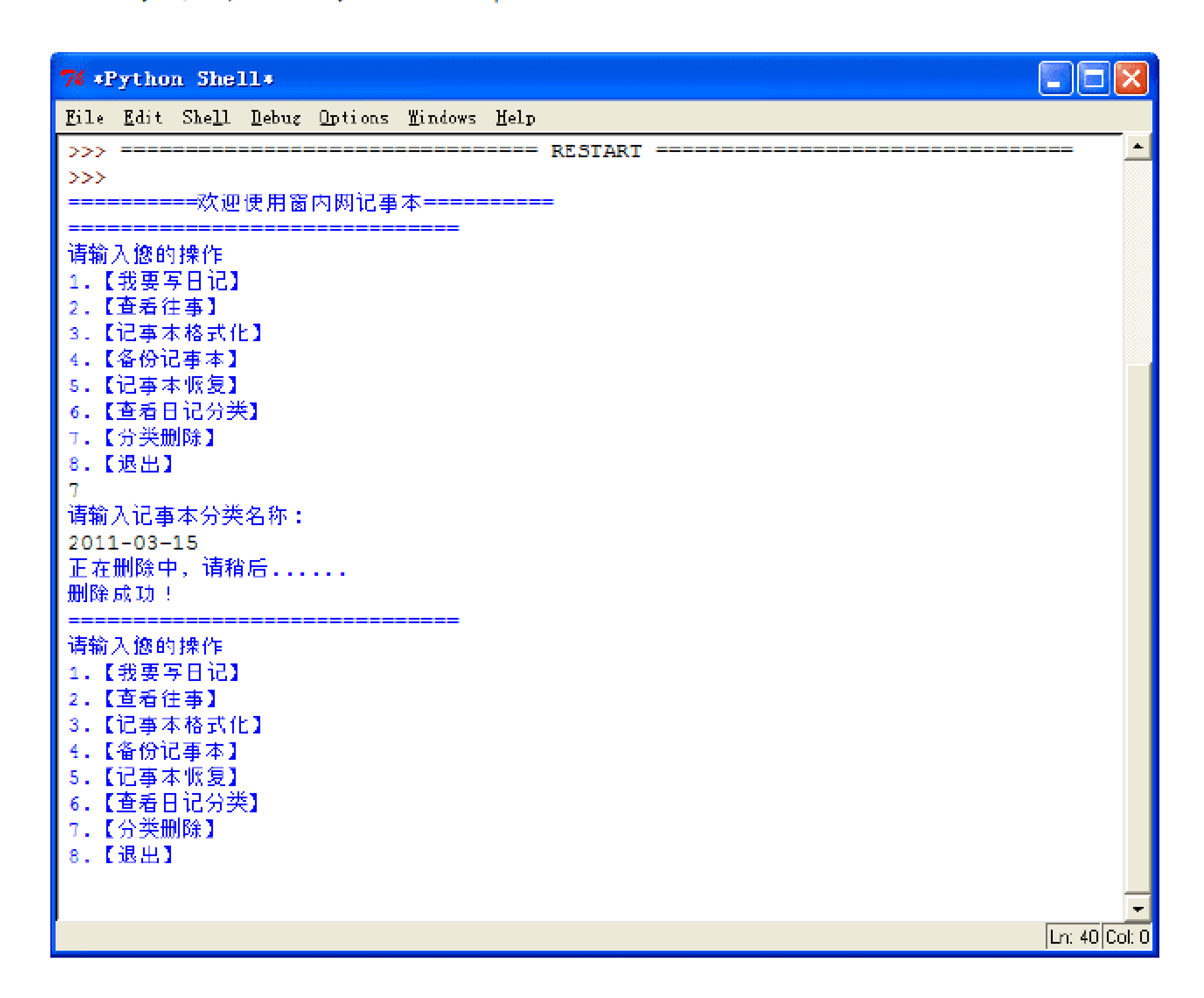


图 8-11　日志分类删除功能



第 9 章　构造可容错的应用程序

内容摘要

在编写程序的时候，程序员通常需要辨别事件的正常过程和异常情况。这类异常事件可能是错误，或者是不希望经常发生的事情。为了能够处理这些异常事件，可以在所有可能发生的地方使用条件语句(比如让程序员检查除法的分母是否为零)。但是，这样做不仅会降低效率，而且还会让程序难以阅读。

Python 提供了解决这个问题的方案，就是异常处理。本章将详细介绍 Python 中的异常处理方式及程序调试。

学习目标

- 掌握使用 try ...except 语句捕捉异常。
- 掌握使用 try ...finally 语句捕捉异常。
- 掌握使用 raise 抛出异常。
- 掌握 assert 语句的使用方法。
- 熟悉程序的调试方法。

9.1　Python中的异常

Python 中的异常及异常处理功能是非常强大的，可向用户准确反馈出现异常的信息。在 Python 中，异常也是对象，可对它进行操作。所有异常都是基类 Exception 的成员，它们都继承自基类 Exception，而且都在 exceptions 模块中定义。Python 自动将所有异常名称放在内建命名空间中，所以程序不必导入 exceptions 模块，即可使用异常。一旦引发而且没有捕捉 SystemExit 异常，程序执行就会终止。如果交互式会话遇到一个未被捕捉的 SystemExit 异常，会话就会终止。本节将介绍异常的基类 Exception。

 **视频教学：** 光盘/videos/09/ Python 中的异常简介.avi　　长度：7 分钟

Python 用异常对象(Exception)来表示异常情况，遇到错误后，则会引发异常。如果异常对象并未被处理或捕捉，程序就会用所谓的回溯(Traceback)终止执行。Exception 类是最常用的异常类，该类包括 StandardError、StopIteration、GeneratorExit、Warning 等异常类。

StandardError 类是 Python 中的错误异常，如果程序出现逻辑错误，则将引发该异常。StandardError 类是所有内联异常的基类，放置在默认的命名空间中，因此使用 IOError、EOFError、ImportError 等类，不需要导入 exceptions 模块。StandardError 类中常见的异常如表 9-1 所示。

表 9-1　StandardError类中常见的异常

异常名称	描　述
ZeroDivisionError	除数为 0 引发的异常
AssertionError	assert 语句失败引发的异常
AttributeError	属性引用、分配错误异常
IOError	I/O 操作引发的异常
OSError	os 模块函数引发的异常
ImportError	导入模块时引发的异常
IndexError	索引操作错误引发的异常
KeyError	使用字典中不存在的 key 值而引发的异常
MemoryError	内存耗尽而引发的异常
NameError	变量名不存在而引发的异常
NotImplementedError	方法没有实现而引发的异常
SyntaxError	语法错误引发的异常
IndentationError	代码缩进错误引发的异常
TabError	空格和制表符混合使用引发的异常
TypeError	使用不合格的类型执行运算引发的异常
ValueError	使用不合格的参数值引发的异常

StopIteration 类判断循环是否执行完毕，如果循环执行完毕，则抛出该异常。GenerationExit 类是由 Generator 函数所引发的异常，当调用方法 close()时触发该异常。Warning 类表示程序中的代码引起的警告。下面来看一段代码。

```
def arrayList (obj,index):
    return obj[index]
# 调用上面已经定义的 arrayList()函数
userList=['0001','0002','0003','0004']
print arrayList(userList,4)
```

在该段代码中，要求 arrayList()函数对列表 obj 中超过元素索引以外的对象进行索引运算，当函数尝试执行 obj[index]时，就会触发异常。Python 会替序列检测到超出边界的索引运算，并通过抛出内置的 IndexError 异常进行报告。运行该段代码，输出结果如下：

```
Traceback (most recent call last):
  File "01.py", line 5, in <module>
    print arrayList(userList,4)
  File "01.py", line 2, in arrayList
    return obj[index]
IndexError: list index out of range
```

因为在代码中没有可以捕捉的异常，所以该异常将会一直向上返回到程序顶层，并启用默认的异常处理器，即打印标准错误消息的处理器。这些消息包括引发的异常以及堆栈跟踪，也就是异常发生时激活的程序行和函数清单。

9.2　实现提示异常信息编号功能

在 Python 中，异常会被错误自动触发，也能由代码触发和捕获，异常由 4 个语句处理，本节将会对它们进行介绍。

 **视频教学**：光盘/videos/09/异常的处理.avi　　　　长度：16 分钟

9.2.1　基础知识——使用try ...except捕捉异常

在交互模式提示符环境外启动的程序中，顶层的默认处理器就是立刻终止程序，导致程序无法正常运行，为了解决这个问题，可以使用 try...except 或 try ...except ...else 语句捕捉异常。

1. 使用try ...except语句捕捉异常

程序中的异常，可以使用 try ...except 语句来捕获，把需要执行的代码放到 try 块中，而把出现异常后所要执行的代码放到 except 块中。当 try 块中的代码出现异常之后，会执行 except 块中的代码，从而捕捉异常。try ...except 结构的格式如下：

```
try:
    <要执行的程序代码>
except <异常对象 1>, <异常信息标识>:
    <当出现异常对象 1 所示的异常时要执行的代码>
except (<异常对象 2,异常对象 3, 异常对象 4,...>), <异常信息标识>:
```

```
    <当出现异常对象 2、异常对象 3、异常对象 4 ...所示的异常时要执行的代码>
...
```

其中，except 语句中可以有多个异常名称，一般多个异常名称用小括号括起来，比如：

```
try:
    s=1/0
except (IndexError , KeyError , ZeroDivisionError) , e:
    print e
```

try ...except 语句的异常处理规则如下：

- 执行 try 块中的语句，如果引发异常，则执行过程会跳到第一个 except 语句中。
- 如果第一个 except 中定义的异常与引发的异常匹配，则执行该 except 中的语句；如果引发的异常与第一个 except 定义的异常不匹配，则会搜索第二个 except 语句。依次类推，直到搜索到与 try 块中引发的异常匹配的 except 语句为止，从而执行对应的 except 块中的代码。
- 允许编写的 except 数量没有限制。为了使代码简略，可以使用一个 except 语句，在该语句中定义异常为 Exception，从而捕捉所有引发的异常。
- 如果所有的 except 都不匹配，则异常会传递到下一个调用本代码的最高层 try 块中。

下面创建一个例子，具体介绍如何使用 try ...except 语句捕捉异常。

```
try:
    s=1/0
except IndexError:
    print 'except'
except KeyError:
    print 'keyerror'
except ZeroDivisionError:
    print 'ZeroDivisionError'
```

在 try 块中的 s=1/0 是可能出现异常的语句，由于被除数为 0，因此 Python 引发异常，程序直接跳转到 except 语句中，首先检测是不是 IndexError 异常，如果不是，则跳转至第二个 except 语句中，检测是不是 KeyError 异常；如果不匹配，则跳转至第三个 except 语句中，检测是不是 ZeroDivisionError。在 try 块中的异常与该 except 中的异常匹配，则执行该 except 块中的代码。运行该段代码，输出结果如下：

```
ZeroDivisionError
```

如果需要捕捉一段程序中所有发生的异常，可以使用空的 except，请看下面的代码。

```
try:
    s=1/0
except:
    print '出现异常'
```

使用空的 except 语句后，无论 try 块中的代码在执行过程中出现哪种异常，都将捕获并执行 except 块中的代码。运行该段代码，输出结果如下：

```
出现异常
```



因为 Exception 是所有异常类的基类，所以可以定义 except 语句中的异常类为 Exception 类，这样定义之后与空的 except 语句达到一样的效果，即都是捕捉了

try 块中的代码所发生的所有异常。其实，except 语句中的异常类默认就是 Exception 类。

2. 使用try ...except ...else语句捕捉异常

使用 try ...except ...else 语句捕捉异常与使用 try ...except 语句捕捉异常类似，只不过多了一个 else 块。当 try 块中的代码没有异常发生时，则跳过 except 块中的代码，执行 else 块中的代码。try ...except ...else 结构的语法格式如下：

```
try:
    <要执行的程序代码>
except <异常对象 1> , <异常信息标识>:
    <当出现异常对象 1 所示的异常时要执行的代码>
except <异常对象 2> , <异常信息标识>:
    <当出现异常对象 2 所示的异常时要执行的代码>
...
else:
    <没有发生异常时要执行的代码>
```

try ...except ...else 与 try ...except 语句的执行规则大同小异，当开始一个 try 语句后，Python 会标识当前的程序环境。当异常出现时，就会回到这里。

try ...except ...else 语句的工作原理如下。

- 如果 try 代码块语句执行时发生异常，Python 就跳回 try，执行第一个匹配该异常的 except 块中的代码。
- 如果异常发生在 try 代码块内，却没有匹配的 except 子句，那么异常就会向上传递到程序中之前进入的 try 中，或者转到这个进程的顶层(这会使 Python 终止当前程序并打印默认的错误消息)。
- 如果 try 块中的代码执行没有异常发生，则 Python 将执行 else 块中的代码。

简而言之，except 分句会捕捉 try 代码块执行时所发生的任何异常，而 else 分句只在 try 代码块执行且不发生异常时才会执行。下面使用 try ...except ...else 来创建一个示例，具体介绍如何使用该语句捕捉异常。

```
num=0
try:
    num=1/1
except IndexError , indexError:
    print indexError
except KeyError , keyError:
    print keyError
except ZeroDivisionError , zeroError:
    print zeroError
else:
    print '运行正常'
    print num
```

运行该程序，首先执行 try 块中的 num=1/1。如果在执行该条语句时，发生异常，则检测该异常是否与 except 语句中的异常匹配，如果匹配，则执行对应的 except 块中的代码；如果执行 try 块中的 num=1/1 没有发生异常，则执行 else 块中的代码。在此段代码中，程序在执行 try 块中的代码时不发生异常，因此直接跳过 except 语句，跳转至 else 语句，执行 else 块中的代码。

运行该段代码，输出结果如下：

```
运行正常
1
```

9.2.2 基础知识——使用try ...finally捕捉异常

try 语句的另一种捕捉异常的形式为特定的形式，和最终结果有关。如果在 try 块中包含了 finally 语句，则 Python 一定会在执行完 try 块中的代码之后再执行 finally 块中的代码(无论 try 块中的代码是否发生异常都将执行 finally 块中的代码)。try ...finally 的语法结构如下：

```
try:
    <要执行的程序代码>
finally:
    <执行完 try 块中的代码之后要执行的代码>
```

使用 try ...finally 语句捕捉异常的工作原理如下。

- 如果执行 try 块中的代码时没有发生异常，则 Python 会跳转至 finally 语句并执行该代码块，然后继续执行程序中 finally 语句之后的代码。
- 如果执行 try 块中的代码时发生异常，则 Python 依然会执行 finally 语句中的代码块，但是接着会将异常向上传递到上层的 try 语句或顶层的默认处理器，程序不会继续执行导致发生异常的语句之后的 try 块中的代码。也就是说，即使发生了异常，finally 代码块还是会被执行。和 except 不同的是，finally 不会终止异常，而是在 finally 代码块执行后，一直处于发生状态。



当确定某些程序代码之后，无论程序的异常行为如何，一段代码必须执行，此时 try ...finally 形式就相当有用了。在实际开发应用中，可以在 finally 块中实现一些清理动作，比如文件的关闭以及断开服务器连接等。

下面编辑一段代码，使用 try ...finally 语句来捕捉异常。下面是一个典型的例子，演示了该语句的典型角色。

```
def fileManager ():
    return file('c:\\a.txt', 'r')
userFile=file('c:\\b.txt','r')
try:
    userFile=fileManager()
    print '打开文件'
finally:
    print '关闭文件'
    userFile.close()
```

在该段代码中，带有 finally 语句的 try 块中包含了一个文件处理函数的调用，以确保无论该函数是否触发异常，该文件都将关闭。在 finally 块中编辑了 userFile.close()的代码。当 try 块中的代码引发异常时，控制流程将跳回，执行 finally 代码块并关闭文件。之后，异常向上传递或者传递至默认的顶层处理器(打印标准错误信息并关闭程序)。当 try 块中的函数引发异常后，在 try 块中的调用函数之后的代码“print '打开文件'”将不会被执行。在该段代码中，

fileManager()函数引发异常，文件 c:\\a.txt 是不存在的，则运行该段代码，输出结果如下：

```
关闭文件
Traceback (most recent call last):
  File "03.py", line 5, in <module>
    userFile=fileManager()
  File "03.py", line 2, in fileManager
    return file('c:\\a.txt', 'r')
IOError: [Errno 2] No such file or directory: 'c:\\a.txt'
```

从结果来看，当执行 try 块中的调用 fileManager()函数的代码时，引发异常，但是并没有将异常立即输出，而是先跳转至 finally 块并执行 finally 块中的代码，执行完之后重新跳回 try 块，将异常传递至默认的顶层处理器，打印出异常详细信息，但并没有执行引发异常语句之后的“print '打开文件'”代码。

9.2.3　基础知识——使用raise抛出异常

在 Python 中，可以使用 raise 语句手工引发异常，其语法格式相当简单。其语法格式如下：

```
raise <异常对象>
```

其中，“异常对象”表示将引发异常的异常名称，且异常名称标识了具体的异常类。

执行 raise 语句，Python 会创建指定异常类的一个对象。raise 语句还可指定对异常对象进行初始化的参数，为此需要在异常类的名称后添加一个逗号以及指定参数(或者由参数构成的一个元组)。一旦执行了 raise 语句，raise 语句后的代码将不能被执行。raise 语句的语法格式如下：

```
raise <异常对象>, <异常信息标识>
```

下面创建一个示例，具体介绍如何使用 raise 语句抛出异常。

```
try:
    raise NameError
except NameError:
    print '抛出一个异常'
```

在该段代码的 try 块中，使用了 raise 语句抛出一个 NameError 异常，然后使用 except 语句捕捉所抛出的 NameError 异常。如果在 try 块中的异常与 except 语句中的异常对象匹配，则执行 except 块中的代码。运行该段代码，输出结果如下：

```
抛出一个异常
```

简单地使用 raise 抛出一个异常，系统报出的异常是什么呢？下面使用 raise 简单方式抛出一个异常。

```
raise ValueError,'invalid argument'
```

运行该段代码，输出结果如下：

```
Traceback (most recent call last):
  File "04.py", line 6, in <module>
    raise ValueError,'invalid argument'
ValueError: invalid argument
```

即抛出的异常类型为 ValueError，异常信息为 invalid argument。

9.2.4 基础知识——自定义异常

Python 允许程序员自定义异常类型，用于描述 Python 异常体系中没有涉及的异常情况。自定义异常必须继承 Exception 类。自定义异常按照命名规范以 Error 结尾，显示地告诉程序员该类是异常类。自定义异常同样可以使用 raise 语句引发，而且只能通过手工方式触发。

自定义异常即自定义数据类型(类)，创建简单的自定义异常类型比较简单，基本语法如下：

```
class exceptionName(baseException):pass
```

基类应该为 Exception 类或继承自 Exception 类。在程序中可以通过创建新的异常类型来命名自定义异常，异常类通常直接或间接地从 Exception 类派生，例如：

```
class MyError(Exception):
    def __init__ (self,value):
        self.value=value
    def __str__ (self):
        return repr(self.value)

try:
    raise MyError(2*2)
except MyError, e:
    print 'My exception occurred, value:', e.value
```

在该段代码中，自定义异常类 MyError，该异常类继承自 Exception 类。在构造函数__init__()中初始化 MyError 的 value 属性，然后定义了输出字符串的方法__str__()。在该方法中将 MyError 类中的 value 属性值返回，最后使用 raise 语句抛出自定义异常 MyError，并传入参数为 2*2，接着使用 except 语句捕捉异常，输出 MyError 类中的 value 属性值。



在异常类中可以定义任何在其他类中定义的东西。为了保持简单，只在其中加入几个属性信息，以供异常处理句柄提取。当在一个新创建的模块中需要抛出几种不同的错误时，通常的做法就是为该模块定义一个异常基类，然后针对不同的错误类型派生出对应的异常子类。

9.2.5 基础知识——使用assert语句

assert 语句用于检测某个条件表达式是否为真。assert 语句又称为断言语句，即 assert 认为检测的表达式永远为真。if 语句中的条件判断都可以使用 assert 语句检测，如果 assert 语句断言失败，则会引发 AssertionError 异常。assert 语句的语法格式如下：

```
assert <条件判断表达式>
```

下面创建一个示例，具体介绍如何使用 assert 语句。

```
username='admin'
password='maxianglin'
assert username == 'admin' and password == 'admin'
```

其实，assert 语句与 if 条件判断语句的使用方法相同。在该段代码中，首先定义了 username 变量的值为 admin，password 变量的值为 maxianglin，而使用 assert 语句检测 username == 'admin' and password == 'admin'表达式中的 password 变量的值却为 admin，因此该表达式的返回结果为 False，从而引起 AssertionError 异常。运行该段代码，输出结果如下：

```
Traceback (most recent call last):
  File "06.py", line 3, in <module>
    assert username == 'admin' and password == 'admin'
AssertionError
```

当出现异常时，将异常信息暴露给用户，而用户并不明白出现此异常的根本原因。一般做法是，将出现该异常的原因告诉用户。assert 语句还可以传递提示信息给 AssertionError 异常。当 assert 语句断言失败时，提示信息将被打印到控制台。请看下面的代码。

```
username='admin'
password='maxianglin'
assert username == 'admin' and password == 'admin','密码错误！'
```

在 assert 语句的表达式之后传入一个提示异常信息的参数，用逗号隔开。运行该段代码，输出结果如下：

```
Traceback (most recent call last):
  File "07.py", line 3, in <module>
    assert username == 'admin' and password == 'admin', '密码错误！'
AssertionError: 密码错误！
```

9.2.6　实例描述

在项目开发过程中，往往需要自定义异常类。当某段代码出现异常时，使用 try ...except 语句捕捉异常，并将异常信息提示给用户。一段代码可能出现多个异常，因此需要使用多个 except 进行捕捉，为了给予用户友好的提示，而不是将出现的一堆异常信息暴露给用户，在程序开发中可以自定义多个类，然后分别给类中的变量赋值，当出现不同的异常时，输出不同类中的不同属性，作为异常的编号。

9.2.7　实例应用

【例 9-1】实现提示异常信息编号的功能。

(1) 新建 myindexerror 模块，即创建 Python 文件，命名为 myindexerror.py。

(2) 在该模块中自定义 MyIndexError 类，在该类中有一个 value 属性，含有一个初始化方法__init__()，在该方法中向属性 value 赋值。MyIndexError 类的代码如下：

```
class MyIndexError:
    def __init__ (self,value):
        self.value=value
```

(3) 新建 myvalueerror 模块，即创建 Python 文件，命名为 myvalueerror.py。

(4) 在该模块中自定义 MyValueError 类，在该类中有一个 value 属性，含有一个初始化方

法__init__()，在该方法中向属性 value 赋值。MyValueError 类的代码如下：

```
class MyValueError:
    def __init__ (self,value):
        self.value=value
```

(5) 新建 Python 文件，命名为 myerror.py。

(6) 在该文件中编辑可执行代码。首先定义一个长度为 4 的列表，然后让用户输入要查询的用户编号，程序根据用户编号在列表中搜索，得到用户所在的位置。也可以让用户输入要查询的用户名，程序根据用户名在列表中搜索，得到该用户的用户编号。同时，程序使用 try ...except ...else 捕捉异常：如果用户所输入的用户编号大于 3，则出现 IndexError 异常，并输出 myindexerror 模块中的 MyIndexError 类中的 value 属性值；如果用户输入的用户名并不在已经定义的列表中，则出现 ValueError 异常，并输出 myvalueerror 模块中的 MyValueError 类中的 value 属性值；如果用户输入的用户编号小于 4 以及所输入的用户名在列表中存在，则输出相应的用户名和用户编号。完整代码如下：

```
import myindexerror
import myvalueerror
userList=['maxianglin','wanglili','malingling','fanxiaoxuan']
user_str=''
user_name=''
input_selectIndex=0
user_name=0
try:
    input_selectIndex=int(raw_input('请输入要查询的用户名编号：'))
    user_str=userList[input_selectIndex]
    input_selectName=raw_input('请输入要查询的用户名：')
    user_name=userList.index(input_selectName)
except IndexError,e:
    print '出现的错误信息编号为：',myindexerror.MyIndexError('1').value
except ValueError,e:
    print '出现的错误信息编号为：',myvalueerror.MyValueError('2').value
else:
    print '您输入的编号为'+str(input_selectIndex)+'的用户为：'+user_str
    print '您输入的用户名'+input_selectName+'在列表中对应的索引为：'+str(user_name)
```

9.2.8　运行结果

运行 myerror.py 文件，程序提示用户输入要查询的用户名编号，当用户输入的用户名编号(用户所在列表中的索引)大于 3 时，则出现 IndexError 异常，输出错误信息编号为 1，如图 9-1 所示。

```
Python Shell
File Edit Shell Debug Options Windows Help
    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.2.4
>>> ================================ RESTART ================================
>>>
请输入要查询的用户名编号：4
出现的错误信息编号为：  1
>>>
Ln: 16 Col: 4
```

图 9-1　当用户编号大于 3 时错误信息编号为 1

重新运行 myerror.py 文件，当用户输入的用户名编号小于 4 而要查询的用户名不在列表中时，输出错误信息编号为 2，如图 9-2 所示。

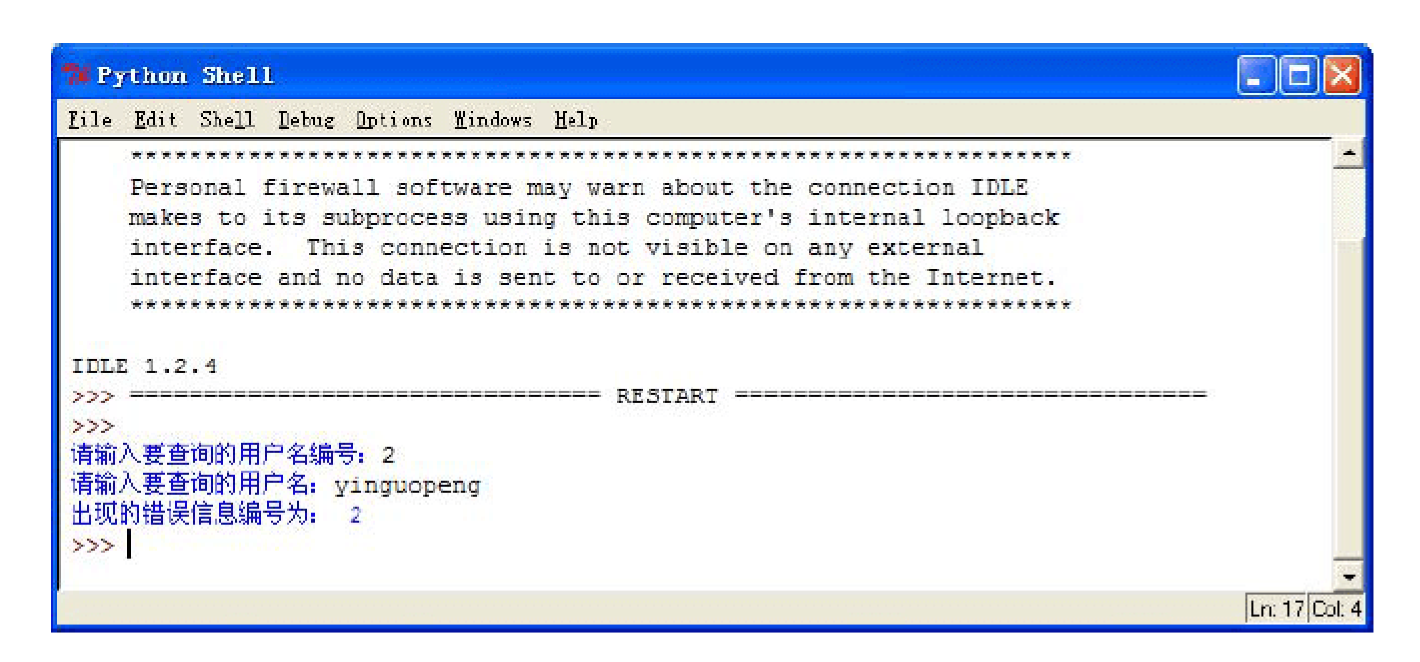


图 9-2　当用户名不在列表中时错误信息编号为 2

再次运行 myerror.py 文件，当用户输入的用户名编号小于 4 且要查询的用户名在列表中存在时，输出相应的用户编号所对应的用户名以及该用户名所对应的用户编号，如图 9-3 所示。



图 9-3　输入正确的运行结果图

9.2.9 实例分析



源码解析

在上面的例子中，在调用 myindexerror 模块中的 MyIndexError 类时，将参数 1 传递过去，即 MyIndexError 类中 value 的值为 1。也就是说，当出现 IndexError 异常时，输出错误信息编号为 1。同理，当出现 ValueError 异常时，输出错误信息编号为 2。

9.3 使用PythonWin调试程序

作为程序员最头痛的一件事就是程序出错而不知错从何来，那么就需要对程序进行调试。所谓程序调试，就是在将编制的程序投入实际运行前，以手工或编译程序等方法进行测试，修正语法错误和逻辑错误的过程。这是保证计算机信息系统正确性的必不可少的步骤。编完计算机程序，必须送入计算机中测试。不论是 Java 语言还是.NET 语言，都有相应的调试工具，Python 语言也不例外，可以使用 PythonWin 调试程序。本节将详细介绍如何使用 PythonWin 实现程序调试。



视频教学： 光盘/videos/09/使用 PythonWin 调试程序.avi　　长度：5 分钟

PythonWin 是一个优秀的 Python 集成开发环境，在许多方面都比 IDE 优秀。例如其文件名称所示，这个工具是针对 Windows 用户的。PythonWin 具有程序调试的功能，使用 PythonWein 实现程序调试分为以下几个步骤。

(1) 打开要调试的文件。选择 File | Open 命令，将要调试的文件在 PythonWin 中打开。

(2) 设置断点。在可能出现错误的代码行设置断点。将鼠标放在需要设置断点的代码行，然后执行 File | Debug 命令，弹出含有许多选项的子菜单，其中 Go 表示开始调试，快捷键为 F5；Step in 表示单步执行，快捷键为 F11；Step out 表示单步跳出，快捷键为 F10；Stop 表示停止调试，快捷键为 Shift+F5；Toggle Breakpoint 表示设置断点，快捷键为 F9。在此，选择 Toggle Breakpoint 选项设置断点，如图 9-4 所示。

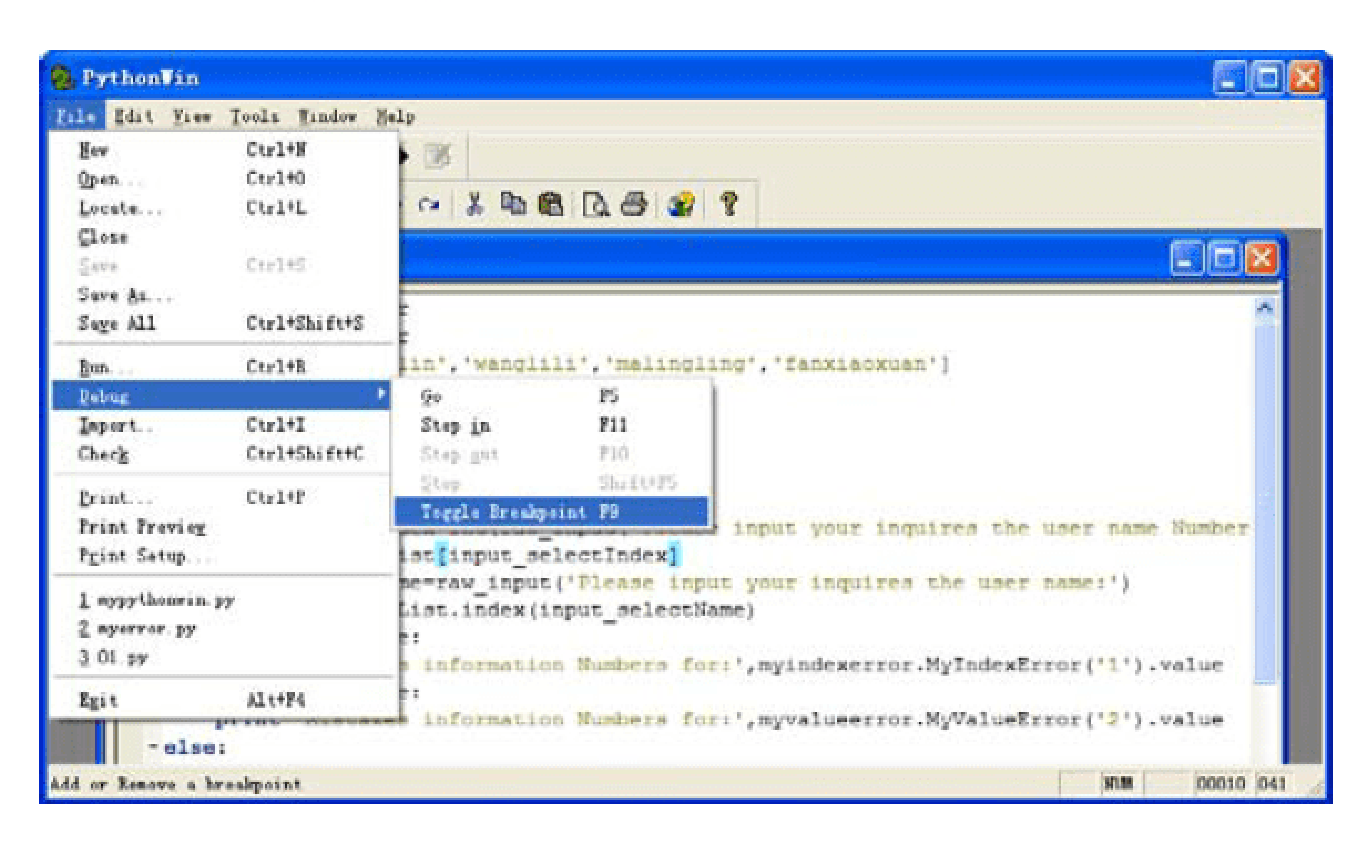


图 9-4 设置断点操作

当设置完所有的断点后，在文件左侧会出现白色的圆圈，如图 9-5 所示。

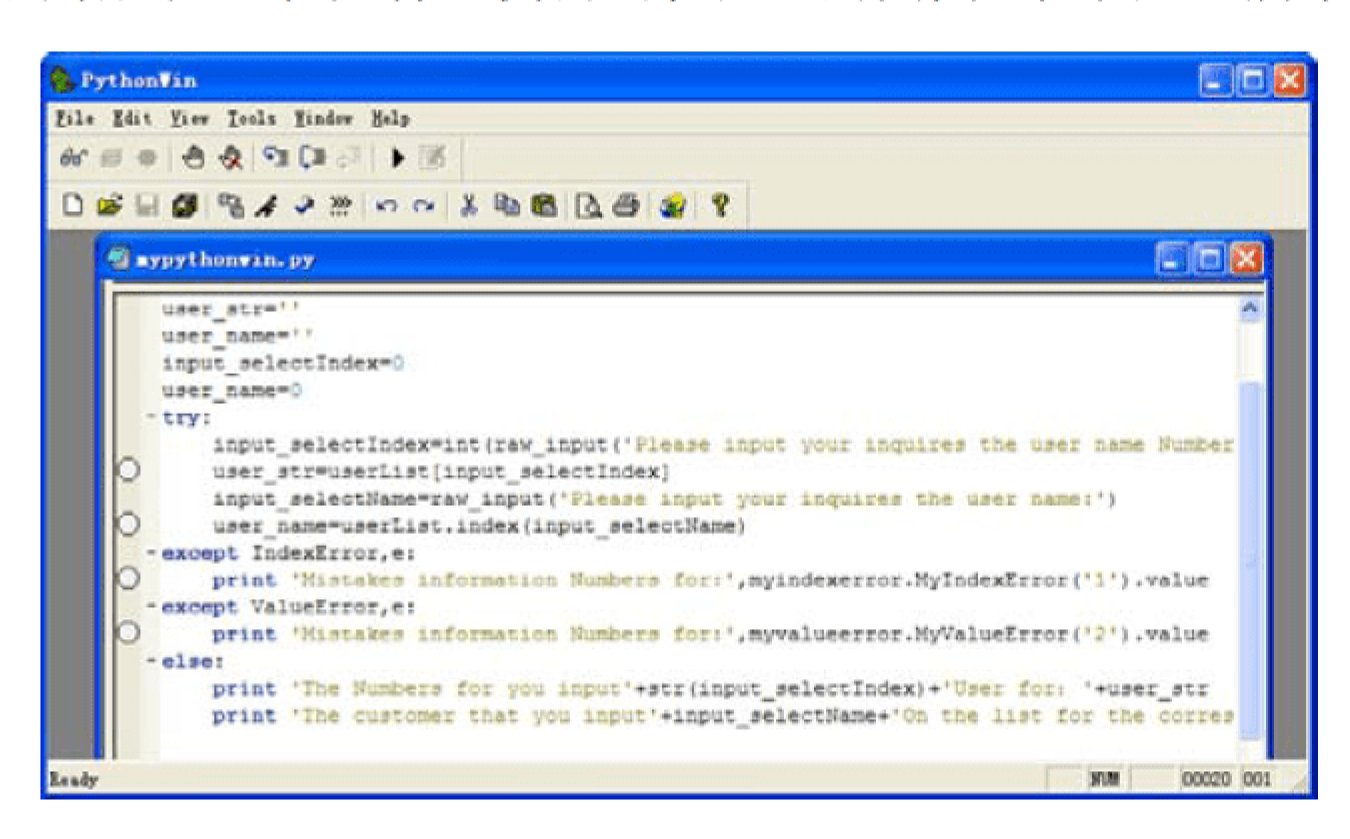


图 9-5　设置断点成功

(3) 设置断点成功之后，按 F5 键启动程序的调试模式，如图 9-6 所示。程序将弹出一个窗口，等待用户输入。

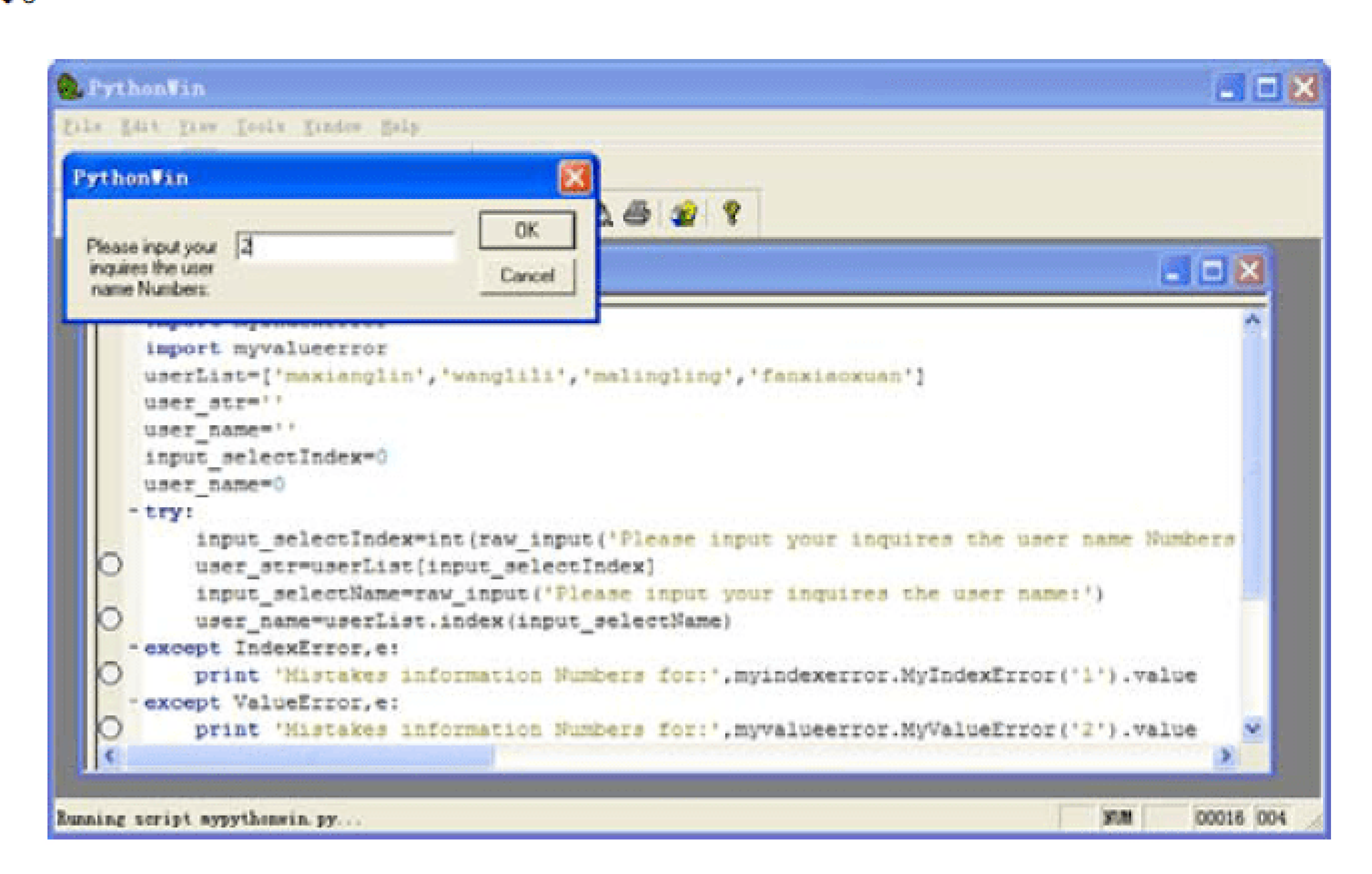


图 9-6　启动程序的调试模式

(4) 输入要查询的用户名编号为 2，然后单击 OK 按钮，调试将从第一个断点处开始执行，如图 9-7 所示。

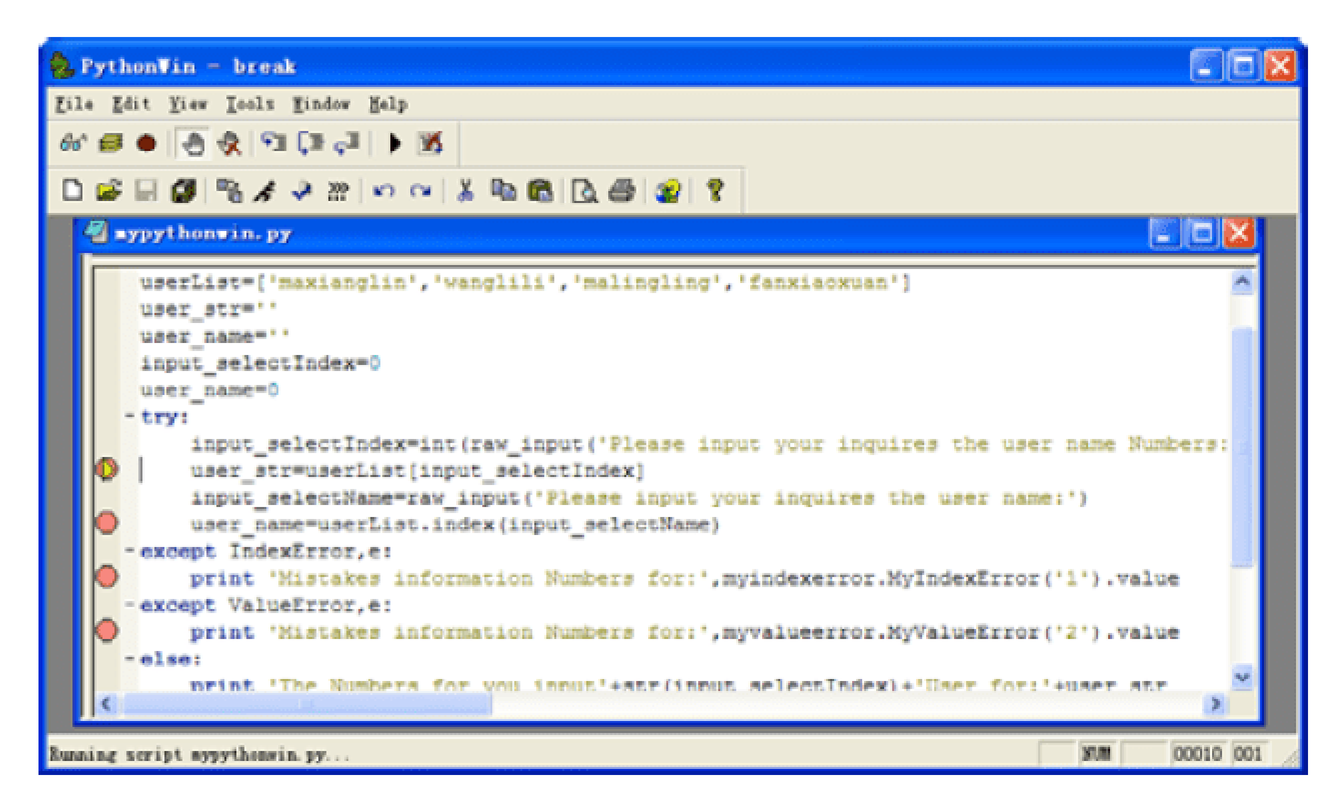


图 9-7　启动程序的单步调试模式

(5) 按 F11 键单步调试程序，当黄色箭头移到 input_selectName=raw_input('Please input your inquires the user name:')语句时，弹出 PythonWin-running 对话框，提示用户输入要查找的用户名，如图 9-8 所示。

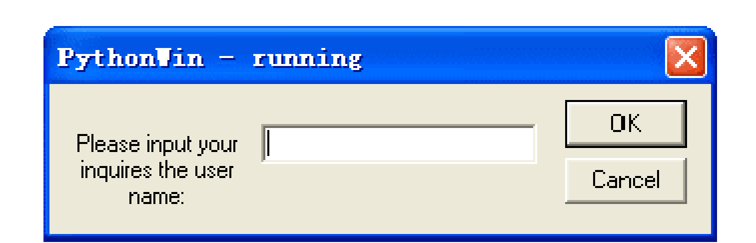


图 9-8　用户名输入对话框

(6) 在弹出的 PythonWin - running 窗口的输入框中输入 yinguopeng，然后单击 OK 按钮，继续执行代码。这时可以在 Interactive Window 窗口中查看程序中各变量的值。当程序执行过 input_selectName=raw_input('Please input your inquires the user name:')代码时，Interactive Window 窗口中提示 ValueError 异常，如图 9-9 所示。

```
Interactive Window
Files\Python\Lib\site-packages\pythonwin\Lib\site-packages\pythonwin\pywin\debugger\debugger.py", line
655, in run
  exec cmd in globals, locals
  File "C:\Documents and Settings\Administrator\D7C0C3E6\09\mypythonwin.py", line
11, in <module>
    input_selectName=raw_input('Please input your inquires the user name:')
  File "D:\Program
Files\Python\Lib\site-packages\pythonwin\Lib\site-packages\pythonwin\pywin\framework\ap
p.py", line 369, in Win32RawInput
    raise KeyboardInterrupt("operation cancelled")
KeyboardInterrupt: operation cancelled
>>> Unhandled exception while debugging...
Traceback (most recent call last):
  File "C:\Documents and Settings\Administrator\D7C0C3E6\09\mypythonwin.py", line
12, in <module>
    user_name=userList.index(input_selectName)
ValueError: list.index(x): x not in list
[Dbg]>>>
```

图 9-9　出现ValueError异常

(7) 当发现出现异常所在的代码行后，按 Shift+F5 停止调试，修改代码，再次运行。



当执行调用其他模块中的类时，按下 F10(单步跳出)，调试程序会跳转到其他模块的类中执行代码，执行完类中的代码后再返回。

9.4　使用Eclipse for Python调试程序

PythonWin 调试程序的操作与常用开发工具的使用习惯不同，在另一个窗口中才能查看程序中变量的值，使用起来不够方便。Eclipse 作为“万能”开发工具支持多种语言，包括 C、Java、PHP、Python 等语言。可喜的是在 Eclipse 开发环境中可以调试 Python 程序。本节将详细介绍如何使用 Eclipse 调试 Python 程序。


视频教学：光盘/videos/09/使用 Eclipse for Python 调试程序.avi　　长度：10 分钟

9.4.1　基础知识——安装PyDev

在调试之前需要在 Eclipse 中新建一个 Python 工程，而新建 Python 工程需要为 Eclipse 安

装 PyDev。安装步骤如下。

(1) 从网上下载 Eclipse for Python 工具包的最新版本。

(2) 解压 Eclipse for Python 工具包，该工具包解压之后包含两个文件夹，分别为 features 和 plugins。将这两个文件夹复制到 Eclipse 安装目录下的 eclipse-SDK-3.5.1-win32/eclipse 目录下，覆盖 Eclipse 安装目录下的 features 和 plugins。

(3) 重新启动 Eclipse，检测 PyDev 是否安装成功。选择 Help | About Eclipse SDK 命令，出现 About Eclipse SDK 窗口，单击该窗口中的 Installation Details 按钮，出现 Eclipse SDK Installation Details 窗口。在该窗口中选中 Installed Software 选项，会在列表项中看到 PyDev for Eclipse 项，如图 9-10 所示。接着选择 About Eclipse SDK 窗口中的 Plug-ins 选项卡，在 Plug-in Id 一栏中至少有 5 个以上分别为 com.python.pydev 和 org.python.pydev 开头的插件，如图 9-11 所示。如果均满足上述两项，则表示安装成功。

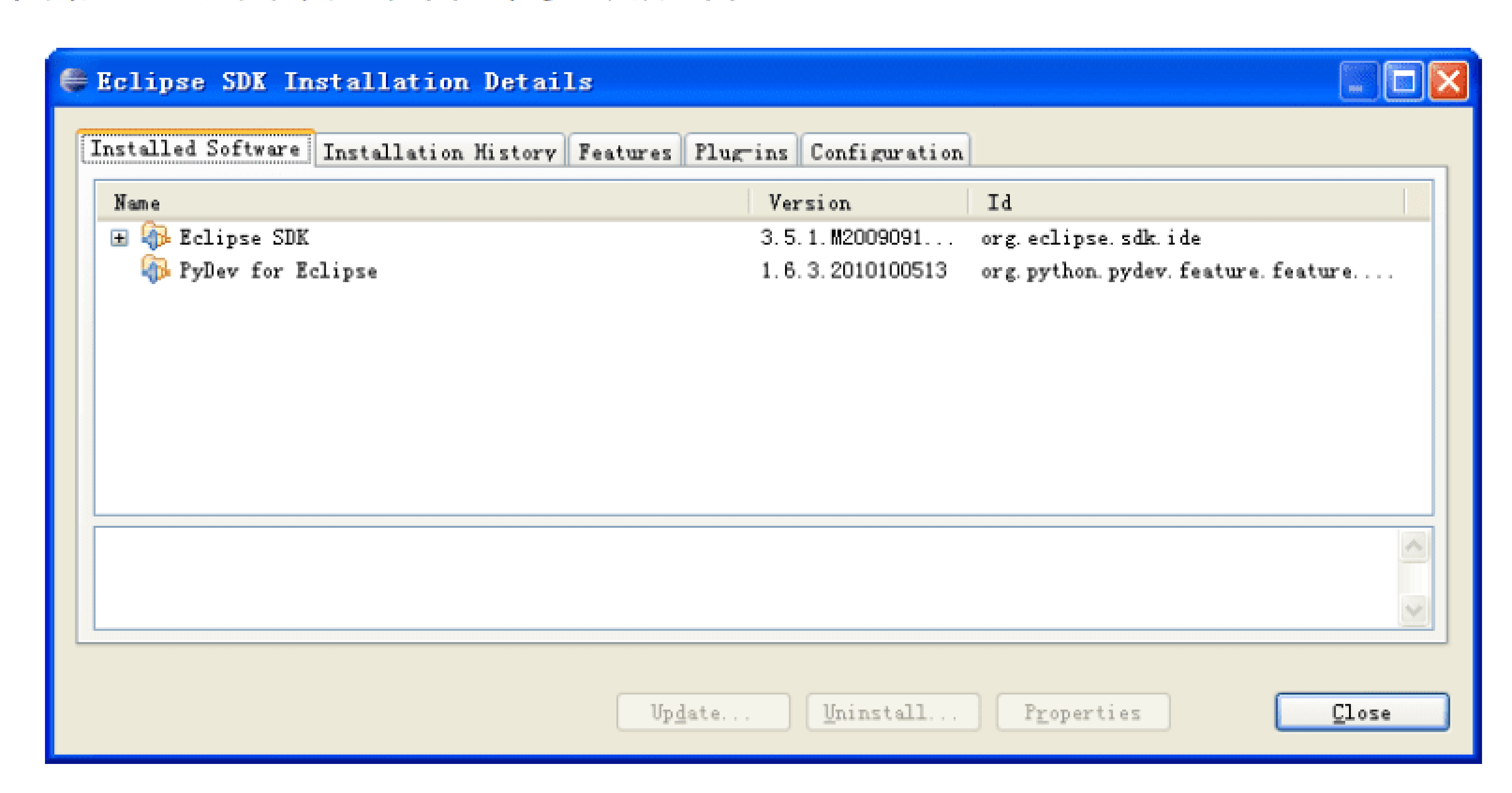


图 9-10 安装软件列表中存在PyDev for Eclipse项

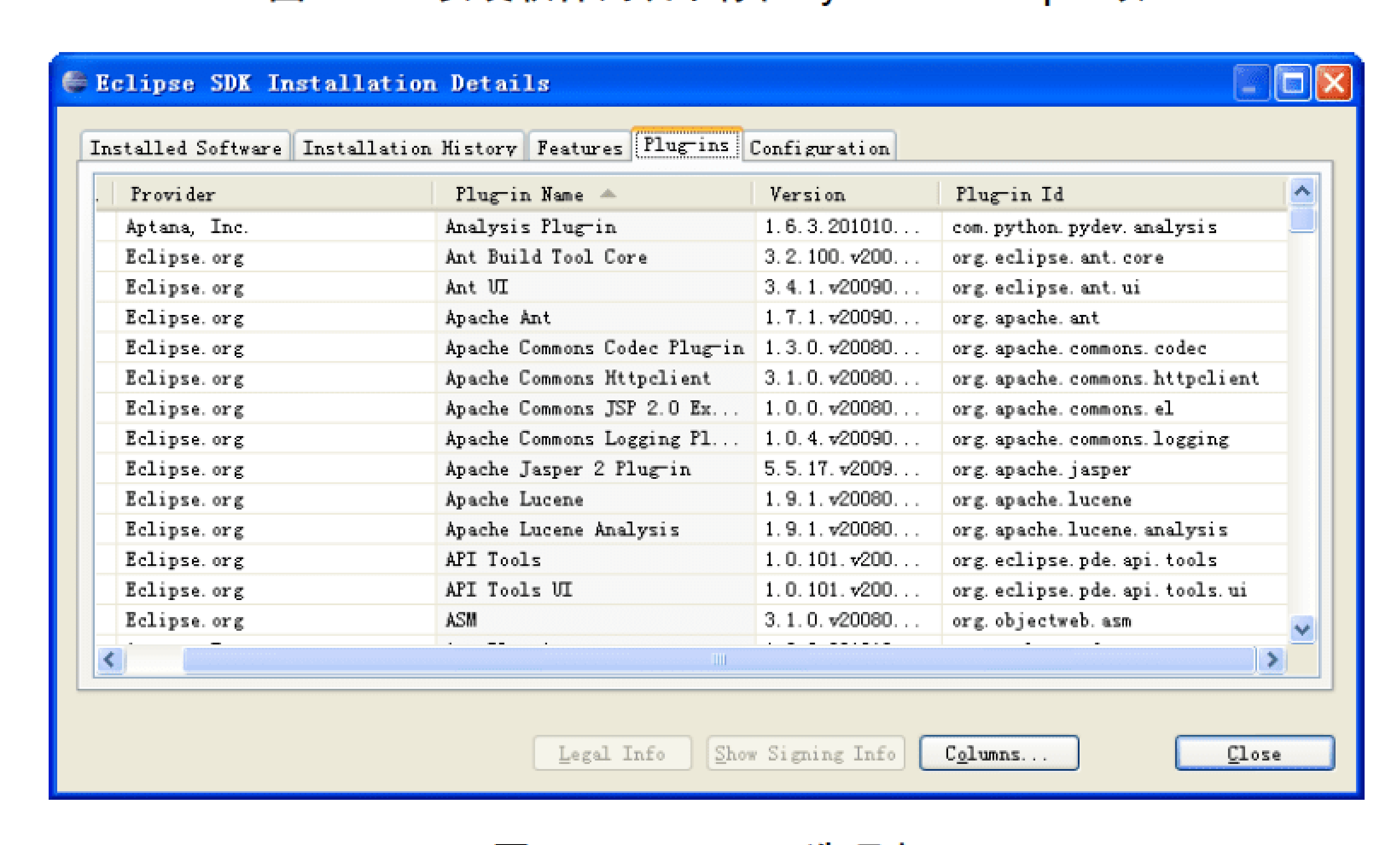


图 9-11 Plug-ins选项卡

9.4.2 基础知识——新建工程

安装 Eclipse 后第一次运行将提示工作空间的路径设置，工作空间就是存放 Eclipse 新建工

程的目录，如图 9-12 所示。

图 9-12　工作空间的路径设置

在这里，选择 Eclipse 新建工程的存放目录为 F 盘下的 workspace 目录，该目录即为 Eclipse 的工程目录。当选择工程的存放目录后，就可以使用 Eclipse 工具新建工程了。

执行 File | New | Pydev Project 命令，弹出 Pydev Project 对话框。在该对话框中可以设置工程的属性，包括工程的名称、存储位置、使用的 Python 标准库等信息。在名称为 Project name 的文本框中输入工程名称 MyPythonPro，在 Project contents 区域中设置工程的存储路径，如果使用默认路径，即 F 盘下的 workspace 目录，则选中 Use default 复选框即可；如果使用其他的存储路径，则单击 Browse 按钮更改工程的存储路径。在 Python type 选项组中选择 Python 选项。在名称为 Grammar Version 的下拉列表中选择 Python 的版本号，这里选择 2.5。选中 Create default 'src' folder and add it to the pythonpath?复选框，表示 Debug 工程创建后会生成一个 src 目录，该目录即为 Python 的源代码目录，如图 9-13 所示。

最后单击 Finish 按钮，完成 Debug 工程的创建，如图 9-14 所示。在 src 目录下新建 Pydev Module 分支，将文件命名为 mypython.py，在该文件中编辑代码以便用于调试。

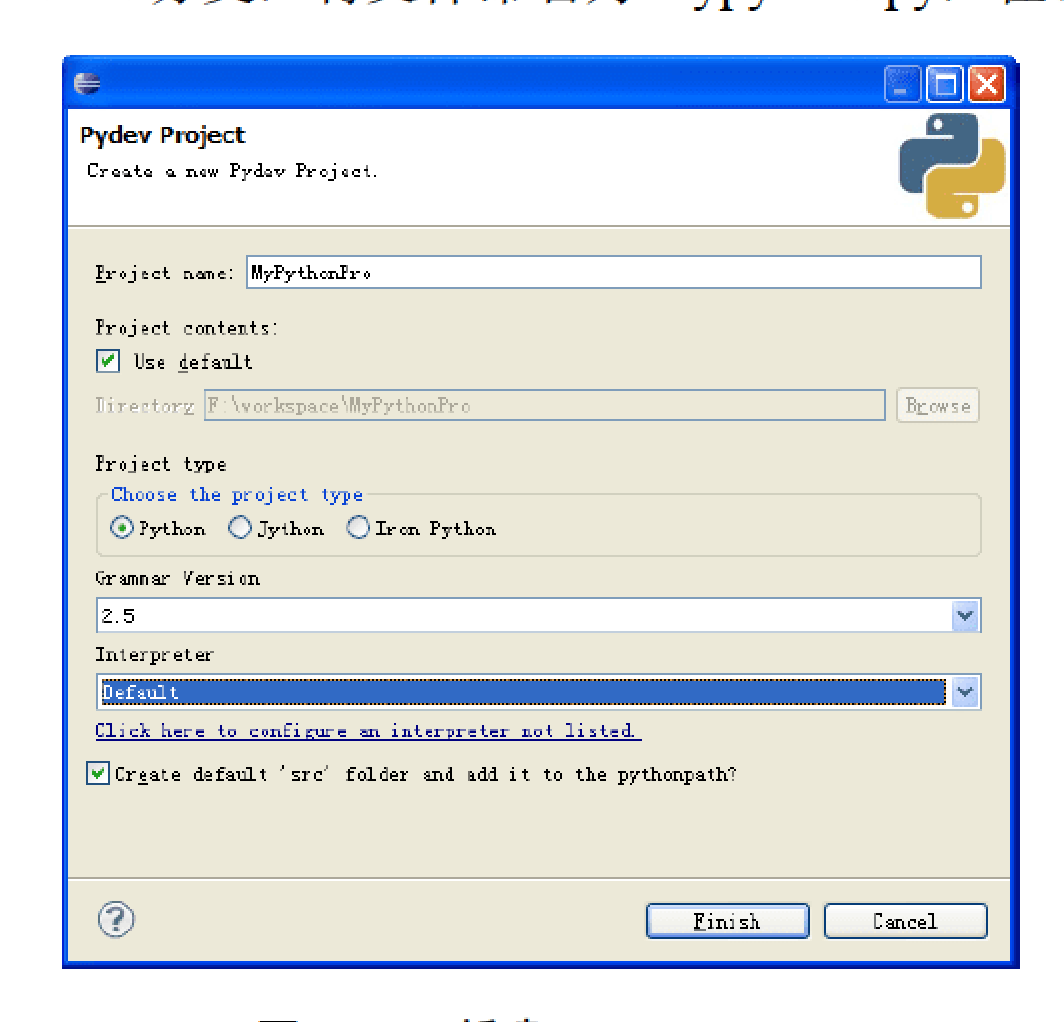


图 9-13　新建Pydev project

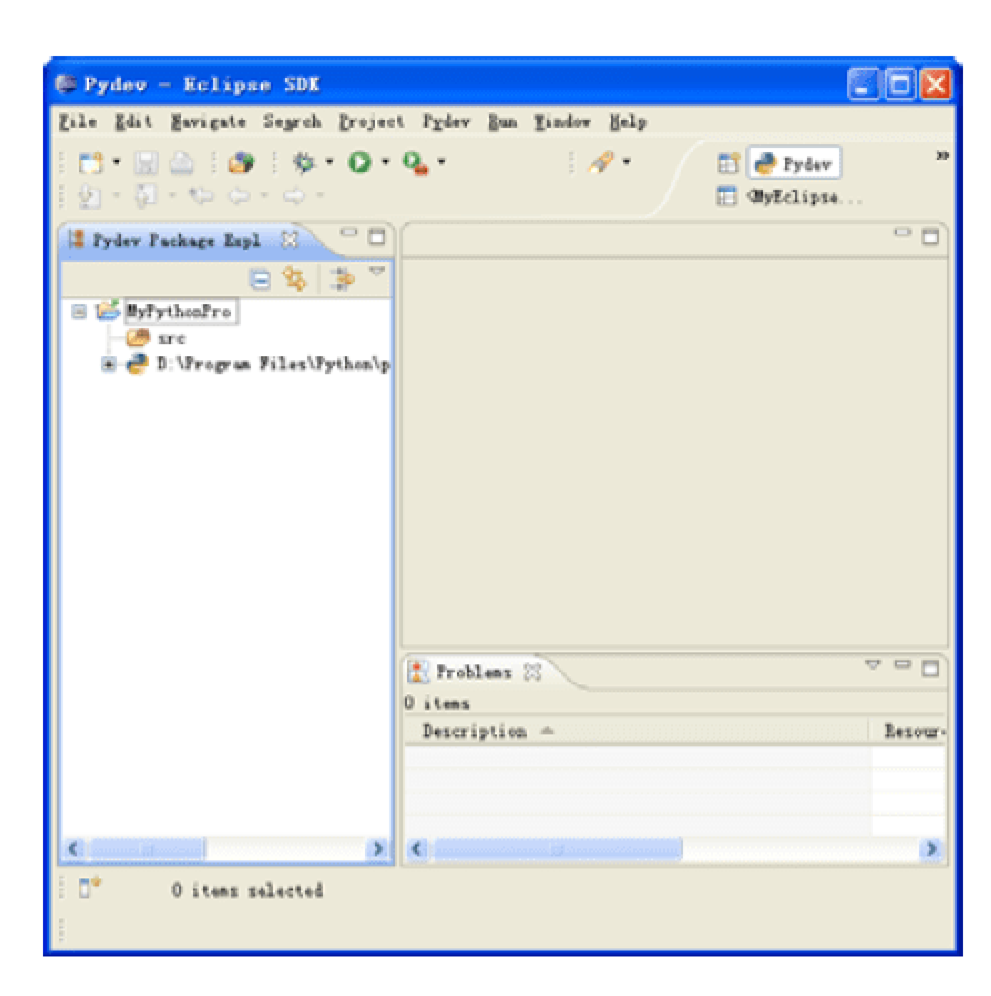


图 9-14　项目创建成功

9.4.3　基础知识——配置调试

在调试程序之前，需要设置 Python 解释器的路径，并导入 Python 环境变量下包含的库文

件。执行 Window | Preferences 命令，弹出图 9-15 所示的 Preferences 窗口，在该窗口中可以对 Eclipse 的开发环境和各种插件进行设置，其中的节点 Pydev 就是 Python 插件的设置项。

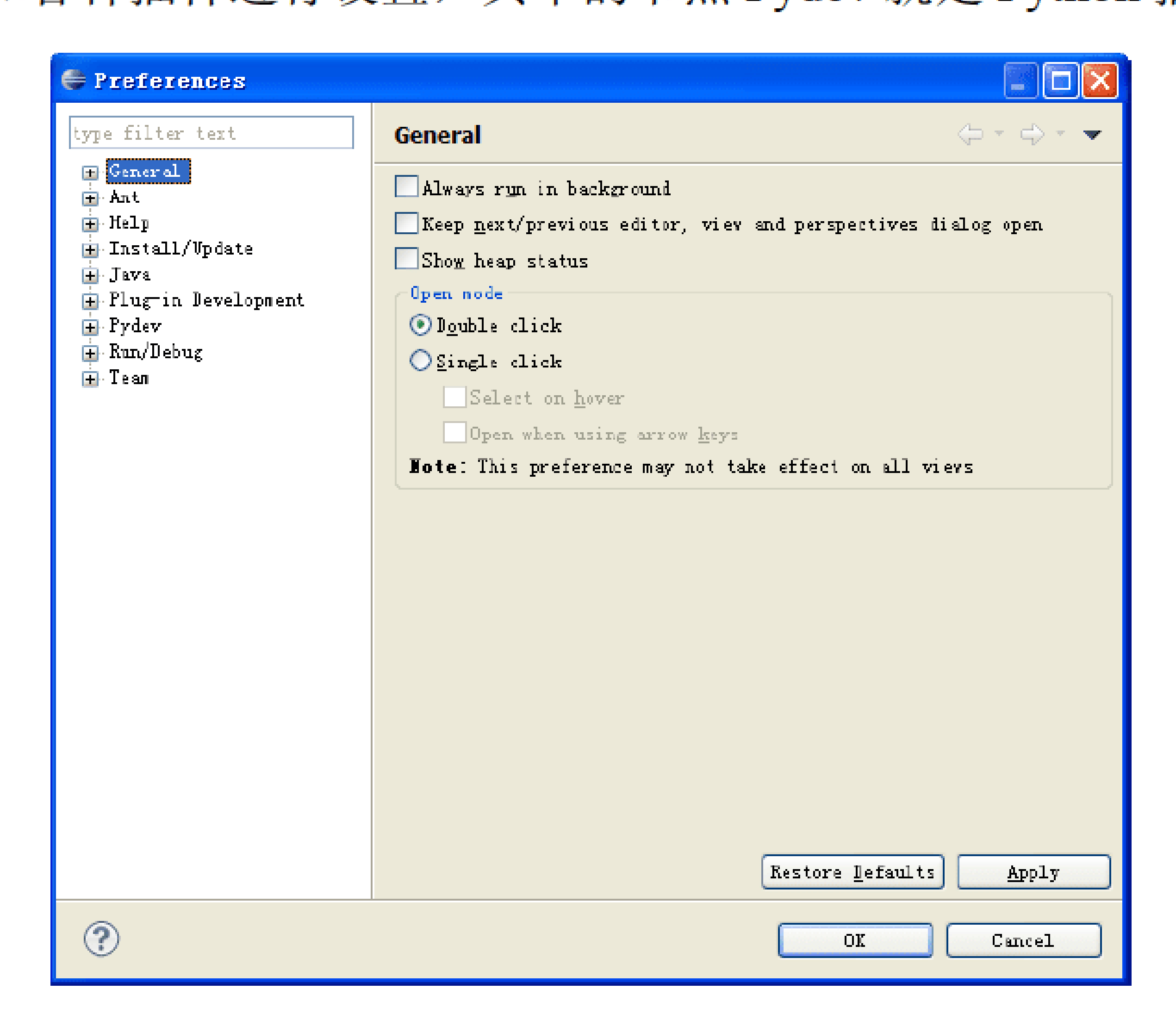


图 9-15　Preferences窗口

展开节点 Pydev 后，选中 Interpreter-Python 子节点，然后单击 New...按钮，加入 python.exe 和 pythonw.exe 所在的路径，如图 9-16 所示。

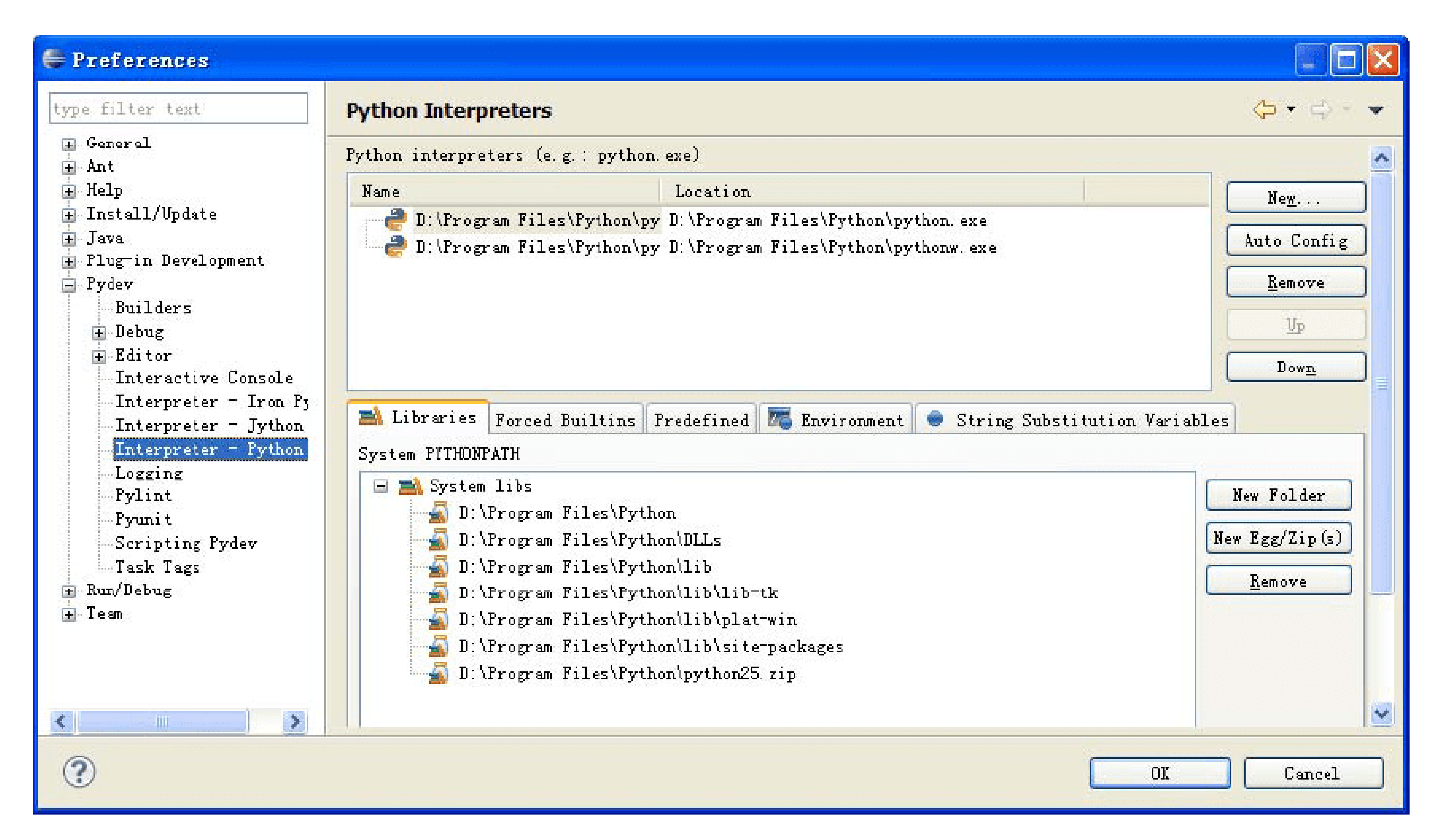


图 9-16　设置Python解释器

接下来，检查一下配置的结果是否正确。在 Libraries 选项卡中检查 System libs 节点，该节点包含 Python 所需的库文件夹所在的路径，如图 9-16 所示。另外，在 Forced Builtins 选项卡页面下，列出了 Python 的内置库，如图 9-17 所示。

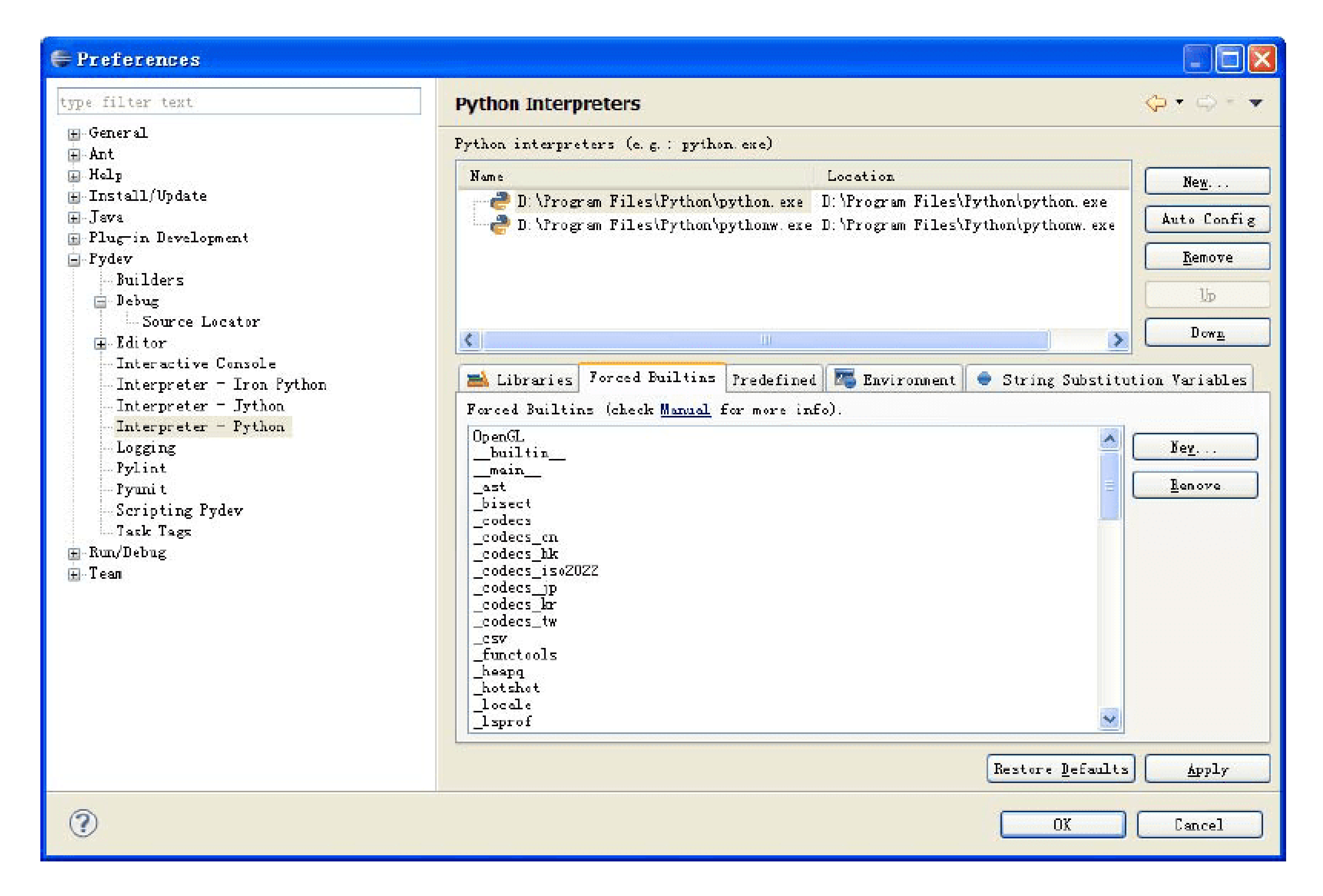


图 9-17　检测是否设置成功

9.4.4　基础知识——设置断点

在刚才创建的文件 mypython.py 中可能出现异常的地方设置断点，这里设置了两个断点。在 Eclipse 工具栏中，单击按钮启动调试程序。当 mypython.py 的调试模式设置成功之后，单击右侧的倒三角形，然后从打开的下拉菜单中选择 MyPythonPro mypython.py 选项，启动调试程序，如图 9-18 所示。

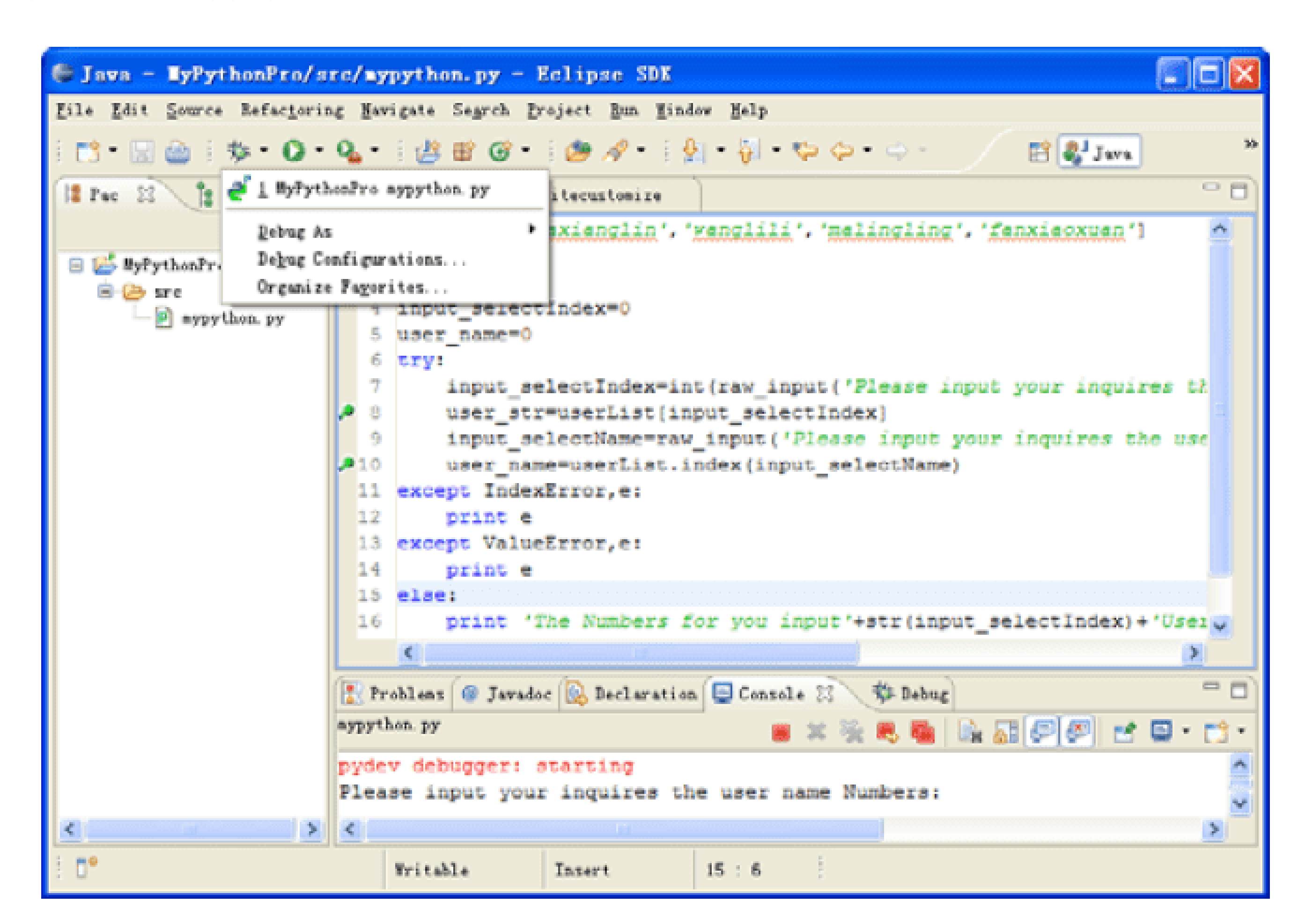


图 9-18　启动调试程序

当启动调试程序之后，系统在控制台(Console)中提示用户输入要查询的用户编号，当用户输入数字 2 之后，按 Enter 键，Eclipse 提示是否进入调试模式，如图 9-19 所示。

单击 Confirm Perspective Switch 对话框中的 Yes 按钮，Eclipse 将自动切换到 Debug 窗口。Breakpoints 标签页显示了当前程序中的断点信息，如图 9-20 所示。

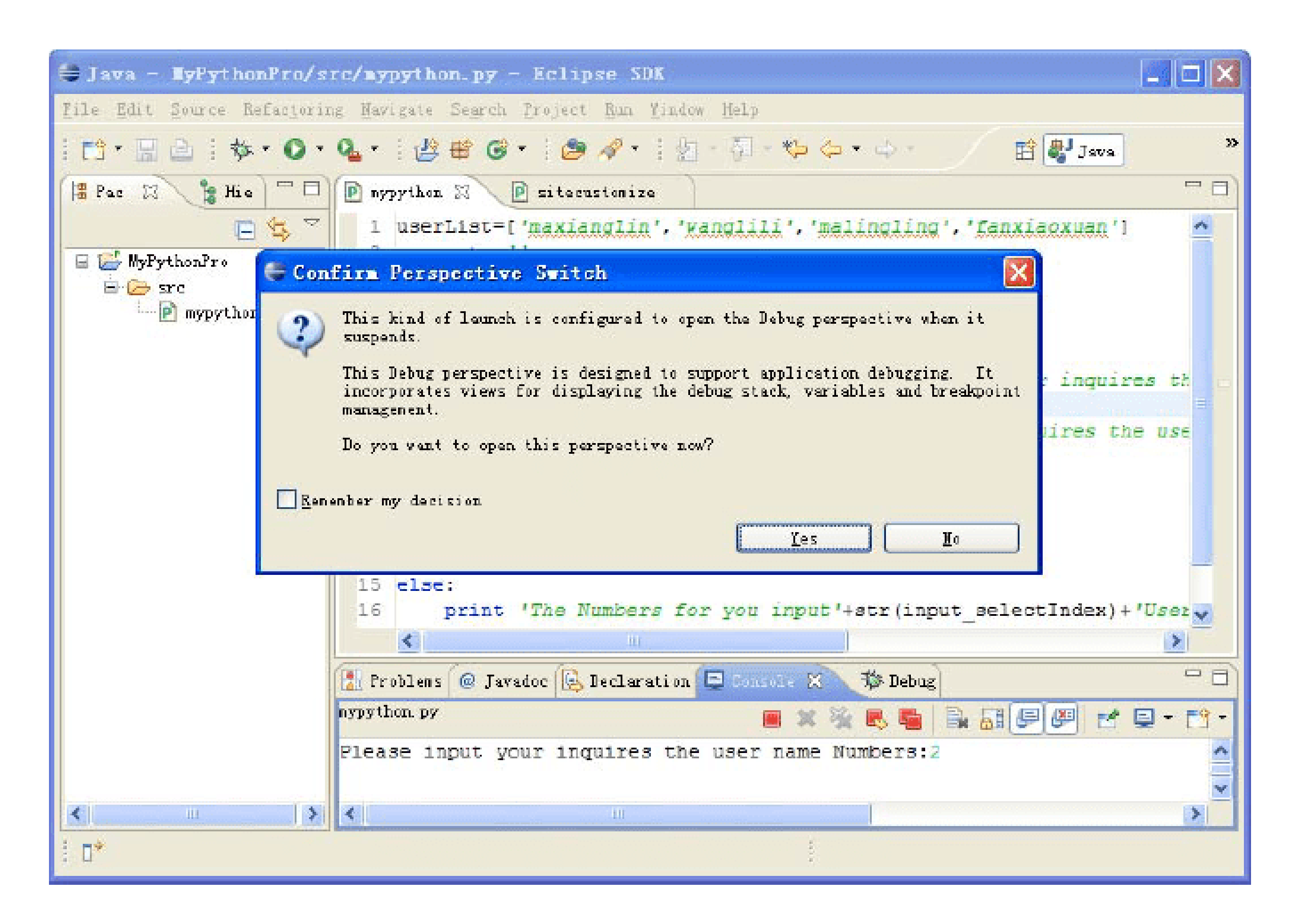


图 9-19　提示用户是否进入调试模式

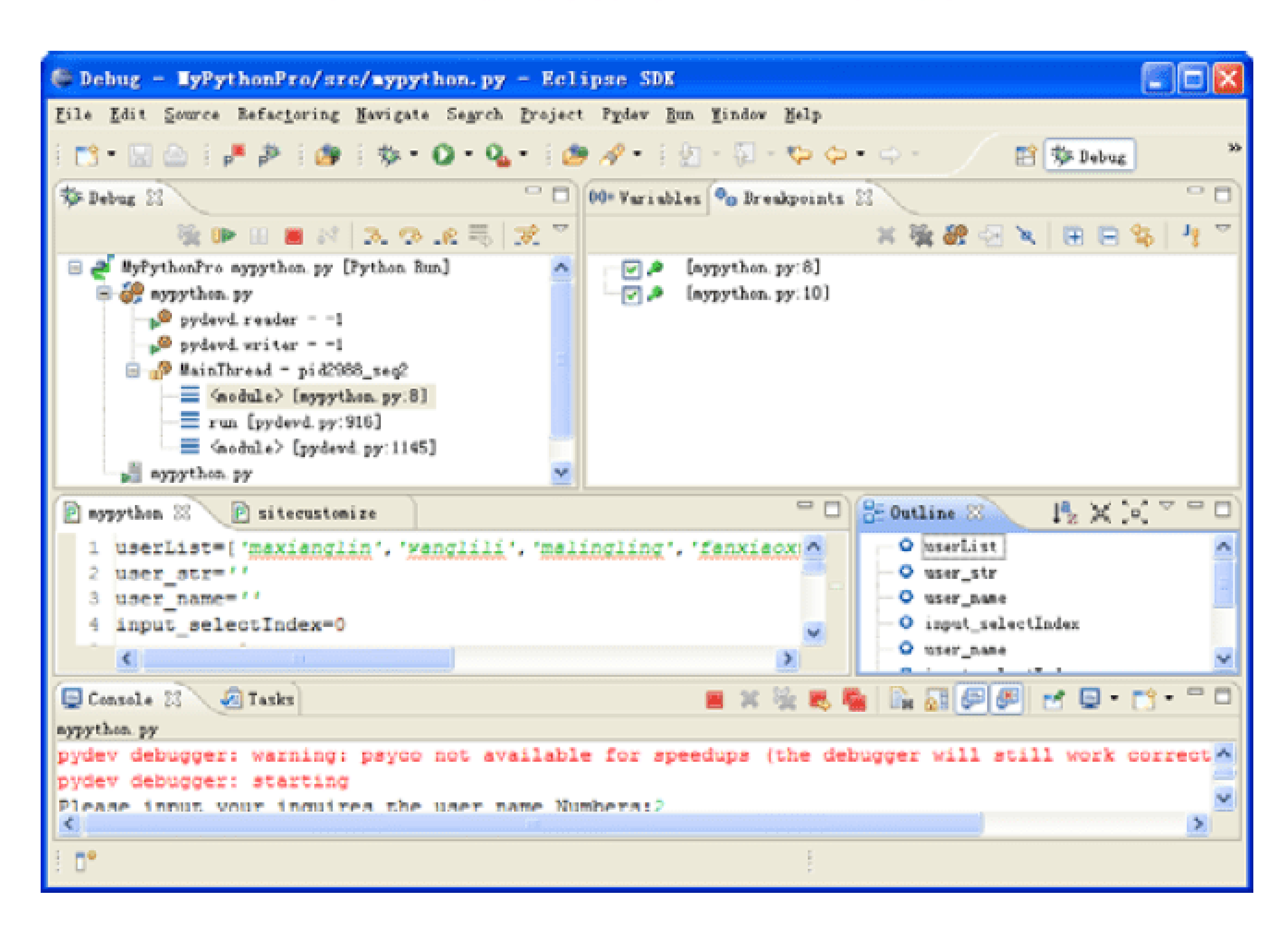


图 9-20　mypython.py的Debug调试窗口

其中，Debug 标签页显示了 mypython.py 的主线程；Outline 标签页显示了当前程序中定义的函数以及所引用的模块名称；Console 标签页用于显示控制台的输出，例如 print 语句的输出和异常。

接着按 F5 键进行单步调试模式，执行 user_str=userList[input_selectIndex]代码，该代码表示通过用户输入的用户名编号获取用户名。在程序中使用 user_str 变量来存储该值。切换到 Variables 标签页，可以查看程序中 user_str 变量的值，如图 9-21 所示。

继续按 F5 键进行单步调试，在控制台将输出 Please input your inquires the user name:等待用户输入要查询的用户名。这时，输入字符串 yinguopeng(该用户名在列表中不存在)并按回车键，继续按 F5 键进行单步执行，当执行到 user_name=userList.index(input_selectName)语句时，出现异常。程序跳出 try 块，转到 except 语句，查找与此异常相匹配的 except 语句，并输出异常。

```
list.index(x): x not in list
```

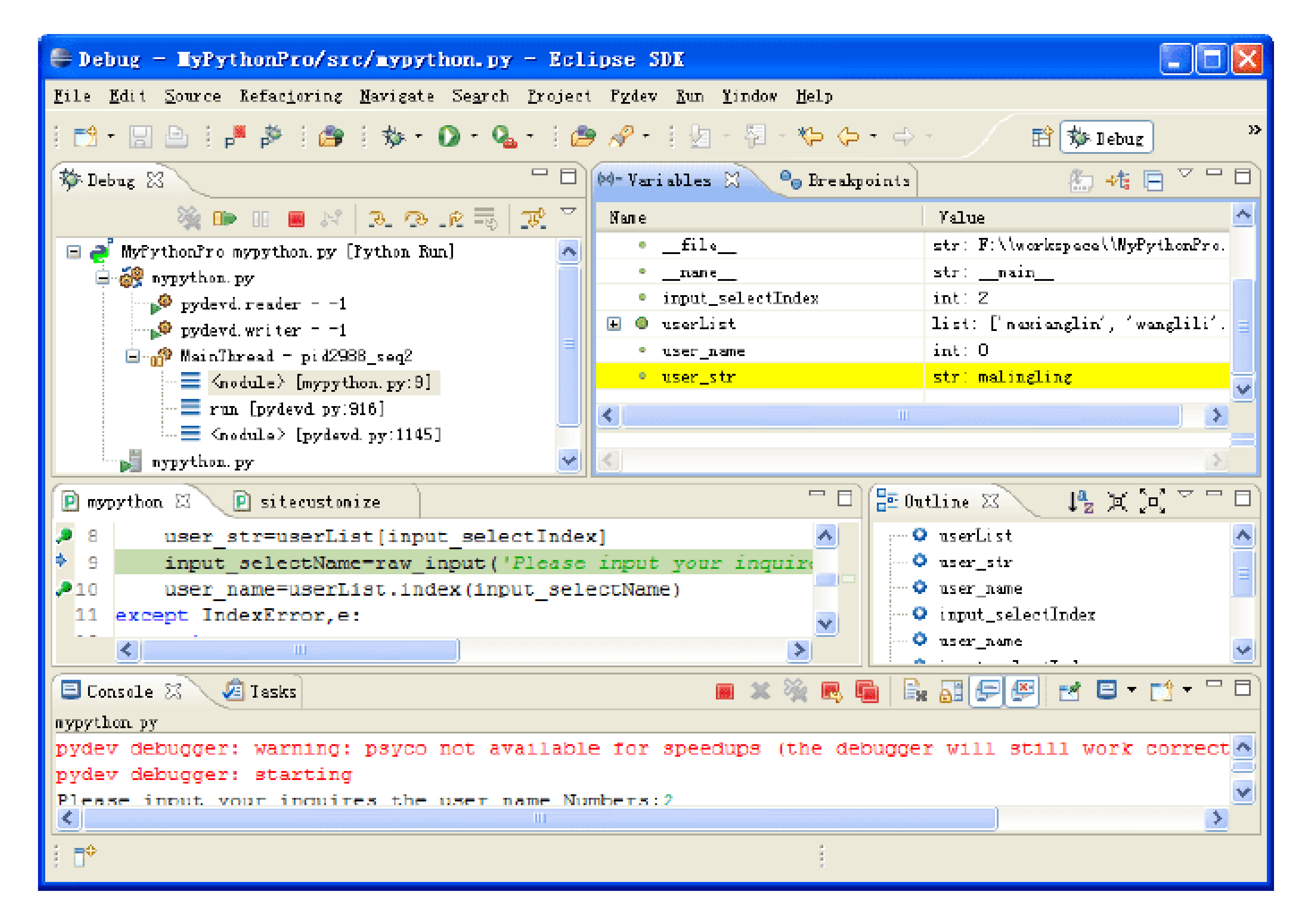


图 9-21　在Variables标签页中查看变量的值

9.5　常见问题解答

9.5.1　常见的捕获异常的方式有哪些



常见的捕获异常的方式有哪些？

网络课堂：http://bbs.itzcn.com/thread-15817-1-1.html

从书上看到捕获异常的方式有很多种，那么最常见的捕获异常的方式有哪些呢？

【解决办法】捕获异常的第一种方式为捕获所有的异常，例如：

```
try:
    a=b
    b=c
except Exception,data:
    print Exception,data
```

运行该段代码，输出结果如下：

```
<type 'exceptions.Exception'> name 'b' is not defined
```

捕获异常的第二种方式为使用 traceback 查看异常，例如：

```
import traceback
try:
    a=b
    b=c
except:
    print traceback.print_exc()
```

运行该段代码，输出结果如下：

```
None
Traceback (most recent call last):
```

```
  File "09.py", line 3, in <module>
    a=b
NameError: name 'b' is not defined
```

捕获异常的第三种方式为采用 sys 模块回溯最后的异常，例如：

```
import sys
try:
    a=b
    b=c
except:
    info=sys.exc_info()
    print info
    print info[0]
    print info[1]
```

运行该段代码，输出结果如下：

```
(<type 'exceptions.NameError'>, NameError("name 'b' is not defined",),
<traceback object at 0x00AFCC60>)
<type 'exceptions.NameError'>
name 'b' is not defined
```

9.5.2　Python的异常体系都有哪些



Python 的异常体系都有哪些？
网络课堂：http://bbs.itzcn.com/thread-15818-1-1.html

最近在研究 Python 语言，Python 语言中有很多捕获异常的方法，那么 Python 的异常体系有哪几种方式呢？

【解决办法】Python 的异常处理可以向用户准确反馈出错信息，所有异常都是基类 Exception 的子类。自定义异常都是从基类 Exception 继承。Python 自动将所有内建的异常放到内建命名空间中，所以程序不必导入 exceptions 模块即可使用异常。下面介绍捕获异常的几种语句结构。

方式一：使用 try ...except 语句来捕获异常，其中可以包含无数个 except 语句来处理异常，如果所有 except 语句都没捕获到则抛出异常到调用此方法的函数内处理，直到系统的主函数来处理。使用 except 子句需要注意的事情，就是使用多个 except 子句截获异常时，如果各个异常类之间具有继承关系，则子类应该写在前面，否则父类将会直接截获子类的异常。放在后面的子类异常就不会被执行。

```
try:
    block
except [excpetion,[data...]]:
    block
except [excpetion,[data...]]:
    block
except [excpetion,[data...]]:
     block
```

方式二：当没有异常发生的时候，执行 else 语句。

```
try:
   block
except [excpetion,[data...]]:
   block
else:
   block
```

方式三：finally 语句。不管有没有发生异常都将执行 finally 语句块。例如，在 python 中打开一个文件进行读写操作，在操作过程中不管是否出现异常，最终都要把该文件关闭。

```
try:
   block
finally:
   block
```

方式四：try ...except ...finally 并用。

```
try:
   block
except:
   block
finally:
   block
```

9.6 习　　题

一、填空题

(1) __________类是 Python 中的错误异常，如果程序中出现逻辑错误，将引发该异常。

(2) __________语句用于检测某个条件表达式是否为真，如果该语句断言失败，会引发 AssertionError 的异常。

(3) 当执行以下代码时，出现的异常类型为__________。

```
try:
    s=1/0
except Exception, e:
    print e
```

二、选择题

(1) 在 Python 中，可以使用__________语句以手工方式引发异常。

A. except　　B. finally　　C. raise　　D. assert

(2) 下段代码输出的结果为：__________。

```
try:
   b=3
   a=b=c
   print a
   print b
   print c
except Exception,e:
   print e
```

A. `name 'c' is not defined`

B. `3 3 3`

C. `name 'a' is not defined name 'c' is not defined`

D. 出现`NameError`类型的异常信息

(3) 自定义异常必须继承自__________类。自定义异常按照命名规范以 Error 结尾，显式地告诉程序员该类是异常类。

A. IndexError　　　　B. ZeroDivisionError

C. MyError　　　　D. Exception

三、上机练习

上机练习： 输入地方名获得当前天气预报。

经常做 Web 开发的程序员应该知道，在网页中实现天气预报功能非常简单，只需要获取该城市对应的天气预报页面，比如郑州的天气预报页面为 http://qq.ip138.com/weather/henan/zhengzhou.htm，然后将该网页生成本地化页面，与要实现天气预报的 Python 文件放在同一目录下，再根据生成的天气预报页面中的标签读取天气预报信息，并显示在页面中。

下面就来模拟一个天气预报功能的实现吧！如果获取到的天气预报页面在生成本地化页面时出现异常，使用 except 捕捉异常。如果没有出错，则根据生成的本地化天气预报页面中的标签读取信息并显示在页面中。程序执行结果如图 9-22 所示。



图 9-22　输入地名获得当前天气预报



第 10 章　持久化的数据

内容摘要

相信做程序开发的人都有这样的意识：在关闭电脑之前，会将项目或者数据保存到硬盘中以备不时之需。这种行为在程序开发中被称为持久化操作。

本章将介绍如何使用持久化模块 dbhash 或者 shelve 模拟小型数据库来读写数据。由于 Python 提供多种数据库连接方法，因此本章还对数据库的连接对象经常使用的方法进行了详细讲解。最后，本章还介绍了 SQLite 数据库的基本语法以及操作方法等方面的内容。

学习目标

- 了解持久化存储的概念。
- 掌握数据库的连接及游标的使用。
- 熟练使用持久化模块读写数据。
- 熟练操作嵌入式数据库 SQLite。

10.1 持 久 化

持久化(Persistence)是将程序数据在持久状态和瞬时状态之间进行转换的机制。持久化主要应用于将内存中的对象存储在关系型数据库中，当然也可以存储在磁盘文件、XML 数据文件中。下面就来认识一下持久化存储。

视频教学：光盘/videos/10/持久化存储.avi　　长度：7 分钟

基础知识——持久化存储

通过上面的介绍，我们了解了持久化的概念，接下来要介绍的是为什么要使用持久化以及持久化在不同语言中是如何实现的。

1. 为什么要使用持久化

你可能会疑惑，为什么要使用持久化机制，而不能永久使用内存呢？这是因为内存与硬盘、磁带、光盘等外存相比过于昂贵，内存的价格要高 2～3 个数量级，而且维护成本也高，至少需要一直供电，所以即使对象不需要永久保存，也会因为内存的容量限制不能一直待在内存中，而需要持久化来缓存到外存。

从另一方面来说，持久化是一种对象服务，就是把内存中的对象保存到外存中，方便以后能够取回。实现数据持久化至少需要实现以下 3 个接口。

- void Save(object o)。把一个对象保存到外存中。
- Object Load(object oid)。通过对象标识从外存中取回对象。
- bool Exists(object oid)。检查外存中是否存在某个对象。

曾经听过这样一句话：凡是可以序列化的对象都可以持久化。的确持久化和序列化这两个概念非常相似，序列化也是一种对象服务，就是将内存中的对象序列化成流或者将流反序列化为对象。总之，需要实现下面两个接口。

- void Serialize(Stream stream,object o)。将对象序列化到流中。
- object Deserialize(Stream stream)。将流反序列化成对象。

有些人将序列化和持久化混为一谈，其实还是有区别的。序列化是为了解决对象的传输问题，传输可以在线程之间、进程之间、内存和外存之间以及主机之间进行。我之所以在这里提到序列化，是因为我们可以利用序列化来辅助持久化，可以说凡是可以序列化的对象都可以持久化，因为序列化相对容易一些(也不是很容易)，所以主流的软件基础设施(比如.NET 和 Java)都已经把序列化的框架完成了。

持久化方案可以分为关系数据库方案、文件方案、对象数据库方案、XML 数据库方案。目前主流的持久化方案是关系数据库方案，关系数据库方案不仅解决了并发问题，更重要的是，关系数据库还提供了持久化服务之外的价值——统计分析功能。刚才我说道，凡是可以序列化的对象都可以持久化，极端地说，我们可以只建立一个表 Object(OID,Bytes)，但基本上没有人这么做，因为一旦这样，我们就失去了关系数据库额外的统计分析功能。

2. 持久化的相关体现

在 Java 语言中，我们使用 Hibernate 框架来完成持久化操作，其中 Hibernate 为应用程序提供了高效的 O/R(Object/Relation)关系映射和查询服务，为面向对象的领域模型到传统的关系型数据库的映射提供了一个使用方便的框架。在 Python 语言中，主要使用一些模块(例如 dbhash、anydbm 和 shelve)完成持久化操作。关于 dbhash、anydbm、shelve 等模块的使用在后面的章节中将会详细介绍。

10.2　Python的数据库支持

众所周知，使用简单的纯文本文件可以实现一些简单的功能。但是想自动支持数据并访问，例如希望让多个用户同时对基于磁盘的数据进行读写，并且不会对该磁盘中的任何文件造成破坏；或者希望使用多个字段或者属性进行大量的复杂搜索等情况，使用标准的数据库是不错的选择。使用数据库处理数据不仅可以处理大量的数据，而且可以使其他程序员容易理解。

在 Java 语言中，我们不仅可以使用 SQL Server 数据库对数据进行操作，还可以使用 Oracle、MySQL 等数据库。Python 语言也不例外，同样支持多种数据库操作。只是在 Python 中，需要一个合适的接口来访问数据库，它就是 DB-API。DB-API 是一个规范，它定义了一系列必需的对象和数据库存取方式，以便为各种各样的底层数据库系统和多种多样的数据库接口程序提供一致的访问接口。接下来介绍的是多种数据库文档在 Python 中使用时所牵涉的相关基础知识。

 **视频教学：** 光盘/videos/10/数据库的连接和游标.avi　　

在 Python 语言中，可以使用的关系类型数据库有 PostgreSQL、MySQL、SQLite 等。为了使用这些数据库，首先需要连接到数据库。在连接数据库时，需要使用 connect 函数。该函数中有多个参数，根据连接的数据库不同而选择不同的参数。表 10-1 列出了 connect 函数常用的参数类型。

表 10-1　connect函数中的常用参数

参 数 名	说　明
dsn	数据源名称
user	连接数据库的用户名
password	连接数据库的密码
host	主机名
database	连接数据库的名称



在后面的章节中，我们会以嵌入式数据库 SQLite 为例，介绍如何使用 connect 函数。

使用 connect 函数返回一个连接对象，该对象表示目前和数据库的会话。接下来就可以使用该对象的方法来对数据库中的数据进行操作了。表 10-2 列出了该连接对象所支持的方法。

表 10-2 connect连接对象的方法

方法名称	说　明
close()	关闭连接之后，连接对象和游标都不能用
commit()	如果支持该方法，就会提交挂起的事务，否则没有任何作用
rollback()	回滚挂起的事务
cursor()	返回一个连接的游标对象

表 10-2 中提到的 commit()方法在程序中总是可用的。也就是说，如果你所连接的数据库不支持事务，那么该方法就没有任何作用。如果该数据库支持事务，该方法就会提交挂起的事务。如果你已经关闭了该数据库的连接，但发现还有未提交的事务，那么该方法会将未提交的事务隐式回滚。

表 10-2 中提到的 rollback()方法，如果数据库支持事务，那么调用该方法可以撤销所有未提交的事务。

如果在程序中调用 cursor()方法，会返回一个游标对象，然后可以通过游标对象执行 SQL 查询并检查结果。游标对象的方法如表 10-3 所示，游标对象的特性如表 10-4 所示。

表 10-3 游标对象的方法

方法名称	说　明
callproc(name[,params])	使用给定的名称和参数调用已经命名的数据库程序，其中参数可有可无
close()	关闭游标
execute(oper[,params])	执行 SQL 操作，参数可有可无
executemany(sql1,sql2)	对每个参数执行 SQL 操作
fetchone()	将查询得到的结果集中的下一行保存为序列
fetchmany([size])	获取查询结果集中的多行
fetchall()	获取结果集中的所有行
nextset()	跳至下一个可用的结果集
setinputsizes(sizes)	为参数预定义一个内存区域
setoutputsize(size)	为获取大量数据的值设定缓存区尺寸

表 10-4 游标对象的特性

特性名称	说　明
description	结果列描述的序列，只读
rowcount	获得结果集中的行数
arraysize	fetchmany 中返回的行数，默认为 1



表 10-4 中介绍的游标对象的方法或者特性有些会在下面讲解，而有些(例如 setinputsizes 或者 setoutputsizes 等)则不会介绍，更多的细节问题需要你查阅资料。

10.3　制作一个可以永久保存的磁盘

记得刚开始接触 Java 时，并不懂得写的程序要随时保存，无奈 MyEclipse 工具有一个缺陷，喜欢在程序写到一半的时候突然关掉，当再次使用该工具打开所写的项目时，里面的代码空空如也，你是不是很着急？唯一能解决的办法就是重新编写代码，然后注意保存。保存后的代码并不会消失，因为你将该代码保存到一个永久的文件中，这个永久的文件是不是和磁盘十分相似，只要不是人为删除，你的代码就不会消失，很方便吧。

在 Java 中，可以使用 Hibernate 框架来进行持久化保存数据。在 Python 中，使用什么方式来将内存中的数据保存到固定的文件系统中呢？下面将作详细介绍。

视频教学：光盘/videos/10/持久化模块.avi　　　　长度：10 分钟

10.3.1　基础知识——持久化模块

在前面的小节中，我们了解了持久化的概念，以及为什么要使用持久化以及使用持久化的好处。那么 Python 中的持久化是如何体现的呢？别急，Python 的标准库为我们提供了几种持久化模块，这些模块可以模拟数据库的操作，将数据保存到指定的文件中。例如 dbhash、shelve、anydbm 等模块。首先来看一下如何使用 dbhash 模块来读写数据。

1. 使用dbhash模块读写数据

上面提到，Python 标准库提供的持久化模块可用于模拟数据库的操作，进而将数据保存到指定的文件中，你是不是很疑惑，这个指定的文件指什么？别急，接着往下看。

这个指定的文件可以是 DBM(Database Management)数据库。DBM 是一种文件式数据库，采用哈希结构进行存储，它并不具备管理能力，但是会比普通文件稳定、可靠，并且查询速度快。不同的操作系统需要使用不同的 Python 模块来实现 DBM 数据库，Windows 系统主要使用 dbhash 模块。dbhash 模块的主要方法就是 open()，其格式如下：

```
open(filename,flag)
```

在上述语法中，filename 表示数据库的名称，flag 表示数据库的打开方式。w 表示读或写数据库，r 表示以只读方式打开数据库，c 表示创建数据库。默认为 r。

下面通过一个例子来说明，代码如下：

```
import dbhash
db=dbhash.open('tem','c')
db['西施']='西施浣纱'
db['貂蝉']='貂蝉拜月'
db['昭君']='昭君出塞'
print'---------没有进行任何操作的数据-----------'
for k,v in db.iteritems():
    print k,v

if db.has_key('西施'):
```

```
    del db['西施']
print '-----------删除键为“西施”对应的数据-----------'
for k,v in db.iteritems():
    print k,v
db.close()
```

在上述代码中，使用 dbhash 模块的 open()方法创建一个名称为 temp 的数据库，其中参数 c 表示如果 temp 数据库不存在，则创建该数据库。由于使用 open()方法返回的 db 对象类似于一个字典对象，因此具有字典的所有属性和方法。接着使用 db 对象的 iteritems()方法遍历 db 中的键和值，并在 for 循环中判断 db 对象中是否存在关键字“西施”，如果存在，则删除对应的数据。输出结果如下：

```
>>>
---------没有进行任何操作的数据-----------
西施 西施浣纱
貂蝉 貂蝉拜月
昭君 昭君出塞
-----------删除键为“西施”对应的数据-----------
貂蝉 貂蝉拜月
昭君 昭君出塞
>>>
```

需要注意的是，dbhash 模块的文件系统仅支持字符串类型的值。下面来看一下为对象 db 赋值的例子，代码如下：

```
import dbhash
db=dbhash.open('tem','c')
db['西施']=1
db['貂蝉']=2
db['昭君']=3
for k,v in db.iteritems():
    print k,v
db.close()
```

运行程序，执行结果如下：

```
>>>

Traceback (most recent call last):
  File "D:\Program Files\Python\d.py", line 3, in <module>
    db['西施']=1
  File "D:\Program Files\Python\lib\bsddb\__init__.py", line 230, in
__setitem__
    _DeadlockWrap(wrapF)  # self.db[key] = value
  File "D:\Program Files\Python\lib\bsddb\dbutils.py", line 62, in DeadlockWrap
    return function(*_args, **_kwargs)
  File "D:\Program Files\Python\lib\bsddb\__init__.py", line 229, in wrapF
    self.db[key] = value
TypeError: Data values must be of type string or None.
>>>
```

从上述结果来看，如果将数值赋给 db 对象，程序将抛出 TypeError 异常类型的错误信息，这的确是一个缺陷。那么有没有既支持字符串又支持数值的模块呢？答案是肯定的。这就是接下来要学习的 shelve 模块。

2. 使用shelve模块读写数据

shelve 模块是 Python 中的持久化对象模块，shelve 模块的使用和 dbhash 模块相似，唯一不同的是 shelve 模块返回的字典类型可以支持 Python 中的基本类型，例如字符串、数字、元组以及列表等。同样，shelve 模块也有 open()方法，其使用格式如下：

```
open(filename)
```

在上述语法中，filename 表示数据库的名称，如果数据库不存在，则创建该数据库。

```
import shelve
db=shelve.open('mydb')
db['dcy']=['dcy','15093077823','shanghai','myworld',4000]
db['ltt']=['ltt','15093077824','beijing','myworld',2500]
db['mxl']=['mxl','15093077825','tianjin','myworld',2000]
db['hoppy']=['hoppy','15093077826','shenzhen','myworld',3500]
print db
db.close()
```

在上述代码中，首先导入 shelve 模块，然后使用 shelve 模块的 open()方法返回一个 shelve 对象 db，接着向 db 对象中添加 4 条记录，之后输出 db 对象中的内容，最后调用 db 对象的 close()方法将数据库连接关闭。执行结果如下：

```
>>>
{'ltt': ['ltt', '15093077824', 'beijing', 'myworld', 2500], 'mxl': ['mxl',
'15093077825', 'tianjin', 'myworld', 2000], '1': ['dcy', '15093077823',
'shanghai', 'myworld', 4000], 'hoppy': ['hoppy', '15093077826', 'shenzhen',
'myworld', 3500], '3': ['mxl', '15093077825', 'tianjin', 'myworld', 2000], '2':
['ltt', '15093077824', 'beijing', 'myworld', 2500], '4': ['hoppy',
'15093077826', 'shenzhen', 'myworld', 3500], 'dcy': ['dcy', '15093077823',
'shanghai', 'myworld', 4000]}
>>>
```



shelve 模块返回字典的 key 值只能是字符串类型，如果是其他类型，则会引发异常。

上面提到了 anydbm 模块，anydbm 模块与 dbhash 模块的使用方法非常相似。下面通过一个例子来说明，代码如下：

```
import anydbm
db = anydbm.open("mydatabase", "c")
db["宝珠"] = "一切是如此的不可思议，如果不是尝过苦涩的滋味，我想我不会发现，你早已悄悄的
在我心中，注入了一口甜蜜，这是触动的感觉"
db["莫莉"] = "我无法想象，如果这世界上没有这股淌入内心清新凉沁的滋味，还会有谁能带给我幸
福的感觉"
db["野倩"] = "品尝着绵密与微甜的结合，让我心中渐渐地有想要呼喊快乐的冲动"
db["佳佳"]="这种缤纷透凉的感觉，让身为女王的我，不爱也难"
db.close()
db = anydbm.open("mydatabase", "r")
print '---------------甜心小姐大赛之为自己最喜欢的糖果口味想一句台词
----------------'
for key ,value in db.iteritems():
    print key+'为最喜欢的糖果口味说的台词是：'+value
```

在上述代码中，首先导入 anydbm 模块，接着使用 anydbm 的 open()方法创建一个名称为

mydatabase 的数据库，并返回一个 anydbm 对象 db。接下来向对象 db 中添加数据，然后调用 db 对象的 close()方法，关闭数据库连接。再次调用 anydbm 模块的 open()方法，其中传入的参数 r 代表打开数据库 mydatabase。最后使用 for 循环将 db 对象中的内容打印输出。执行的结果如下：

```
>>>
---------------甜心小姐大赛之为自己最喜欢的糖果口味想一句台词---------------
宝珠为最喜欢的糖果口味说的台词是：一切是如此的不可思议，如果不是尝过苦涩的滋味，我想我不会发现，你早已悄悄地在我心中，注入了一口甜蜜，这是触动的感觉
莫莉为最喜欢的糖果口味说的台词是：我无法想象，如果这世界上没有这股淌入内心清新凉沁的滋味，还会有谁能带给我幸福的感觉
野倩为最喜欢的糖果口味说的台词是：品尝着绵密与微甜的结合，让我心中渐渐地有想要呼喊快乐的冲动
佳佳为最喜欢的糖果口味说的台词是：这种缤纷透凉的感觉，让身为女王的我，不爱也难
>>>
```

10.3.2 实例描述

我比较喜欢有趣的文章，因此在闲来无事的时候喜欢搜寻有趣的博客、说说等。今天早上，正当我看得津津有味的时候，突然插头被我碰了一下，机器被关机了。当再次打开的时候，我怎么也找不到那篇文章了，很想大喊一声：如果再给我一次机会，一定老老实实地将那篇文章看完，绝不乱碰。哎，与其抱怨，还不如想一个能解决这类问题的好方法呢，以免类似的情况再次发生。

这不，我制定了一个文件系统，既可以将第一眼看到的内容永久保存到一个数据库中，又可以根据文章的标题将存储的内容删除，一举两得。这下就不用担心文章找不到了。下面来看一下我的实现思路。

10.3.3 实例应用

【例 10-1】制作一个可以永久保存的磁盘。

(1) 创建一个名称为 save.py 的文件。

(2) 在 save.py 文件中添加代码。

```
import shelve
class Person:
    def __init__(self,title,content):
        self.title = title
        self.content = content
    def say(self):
        print '您输入的标题是:%s \t 您输入的内容是:%s' % (self.content,self.content)
temppath = 'MyGood'
def init():
    m = {}
    f = shelve.open(temppath,'w')
    f["init"] = "--------------欢迎您使用万能记事本---------------"
    f.close()
init()
```

```
print '请选择您的下一步:\n(add) 添加永久保存的内容\t(del) 删除永久的内容\t(quit)关闭记事本\n(show) 展示永久保存的内容'
while (True):
    check=raw_input('选择您下一步的操作:')
    if check == 'quit':
        break
    if check == 'add':
        print '-----------欢迎您使用添加主题功能------------'
        titleSave=raw_input('请输入您想永久保存的标题:')
        contentSave=raw_input('请输入您要永久保存的内容:')
        f = shelve.open(temppath,'w')
        f["title"] = titleSave
        f["content"] = contentSave
        print '我添加的标题是：'+f["title"],'我添加的内容是：'+f["content"]
    if check == 'del':
        print '-----------欢迎您使用删除主题功能------------'
        titleDel=raw_input('请输入您想删除标题的键值:')
        f = shelve.open(temppath,'w')
        if f.has_key(titleDel):
            del f[titleDel]
        print '删除成功！'
    if check == 'say':
        titleSave=raw_input('请输入您想永久保存的标题:')
        contentSave=raw_input('请输入您要永久保存的内容:')
        Person(raw_input(titleSave,contentSave))
        Person(raw_input(titleSave,contentSave)).say()
        print "仔细看清楚啊，我没有保存哦"
    if check == 'show':
        f = shelve.open(temppath,'w')
        print '------------------下面是您永久保存的内容------------------'
        for key,value in f.iteritems():
            print key,value
        print '------------------ over ------------------'
        f.close()
```

(3) 保存修改好的代码。

10.3.4　运行结果

运行程序，执行结果如下：

```
>>>
请选择您的下一步:
(add) 添加永久保存的内容    (del) 删除永久的内容    (quit)关闭记事本
(show) 展示永久保存的内容
选择您下一步的操作:
```

当你输入的操作为 add 时，执行结果如下：

```
-----------欢迎您使用添加主题功能------------
请输入您想永久保存的标题:信仰
请输入您要永久保存的内容:你是我一生的信仰
我添加的标题是：信仰 我添加的内容是：你是我一生的信仰
选择您下一步的操作:
```

当你输入的操作为 show 时，显示结果如下：

```
------------------下面是您永久保存的内容------------------
init --------------欢迎您使用万能记事本---------------
title 信仰
content 你是我一生的信仰
------------------ over ------------------
选择您下一步的操作:
```

当您输入的操作为 quit 时，执行结果如下：

```
选择您下一步的操作:quit
>>>
```

10.3.5　实例分析



源码解析

在本实例中，主要使用了 shelve 的 open()方法来创建一个 temppath 数据库，并返回一个 shelve 模块对象 f。接着使用 while 循环，只有当用户输入的操作为 quit 时，才跳出循环，否则会根据不同的选择执行不同的操作。

10.4　SQLite数据库的使用

在学习 Java 的时候，以使用 SQL Server 数据库为主，当然也可以使用 Oracle、Access 或者 MySQL 数据库。无论使用哪种数据库，在做网站或者系统时都少不了对数据库中的数据进行增、删、改、查等操作。但对嵌入式数据库 SQLite 来说还是第一次听到，什么是 SQLite 数据库呢？在编程语言中如何进行操作呢？想必你对 SQLite 数据库的一切都充满了好奇。接下来我们看一下在 Python 语言中如何对 SQLite 数据库进行操作。

**视频教学：**光盘/videos/10/嵌入式数据库 SQLite.avi　长度：6 分钟

10.4.1　基础知识——嵌入式数据库SQLite

SQLite 是一个开源的嵌入式数据库引擎，可应用于 Windows、Linux 等多种操作系统。SQLite 实现了可配置、事务、管理、数据文件的跨平台等特性，目前已经成为应用最多的一种数据系统。接下来我们看一下如何使用 SQLite 数据库。

SQLite 是非常著名的开源嵌入式数据库，可以嵌入到应用程序，并且提供了 SQL 接口来访问数据。SQLite 数据库提供了 DOS 命令行工具，如果你的某个表的数据量比较大，使用 DOS 命令查看结果集就会比较困难，该怎么办？我们可以使用下面的方法。

不知道你有没有遇到过这样的情况：当我们安装成功 SQL Server 数据库软件时，却意外地发现没有对数据库进行操作的管理器。此时，最好的做法就是从网上下载一个 SQL 管理器。同样，SQLite 也有许多第三方的数据库客户端工具用于对数据库对象进行操作，例如 SQLiteSpy 或者 SQLiteManager。

这里我们以 SQLiteManager 管理工具为例，先从 http://www.sqlabs.net/sqlitemanager.php 下载 SQLiteManager 工具。SQLiteManager 对数据库对象支持查询操作、查询分析以及修改表结构等功能。

下载完成之后，对 SQLiteManager 进行安装，其安装过程在这里不再详细介绍。运行 SQLiteManager，出现启动画面，要求注册或购买，如图 10-1 所示。

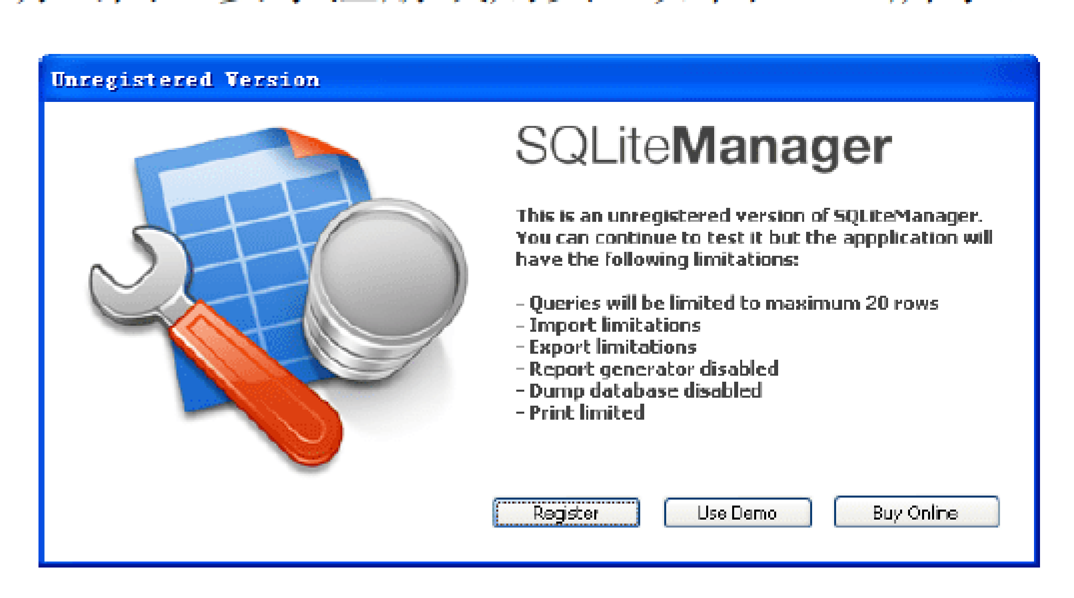


图 10-1　SQLiteManager启动画面

单击 Use Demo 按钮，出现选择数据库对象的界面，如图 10-2 所示。

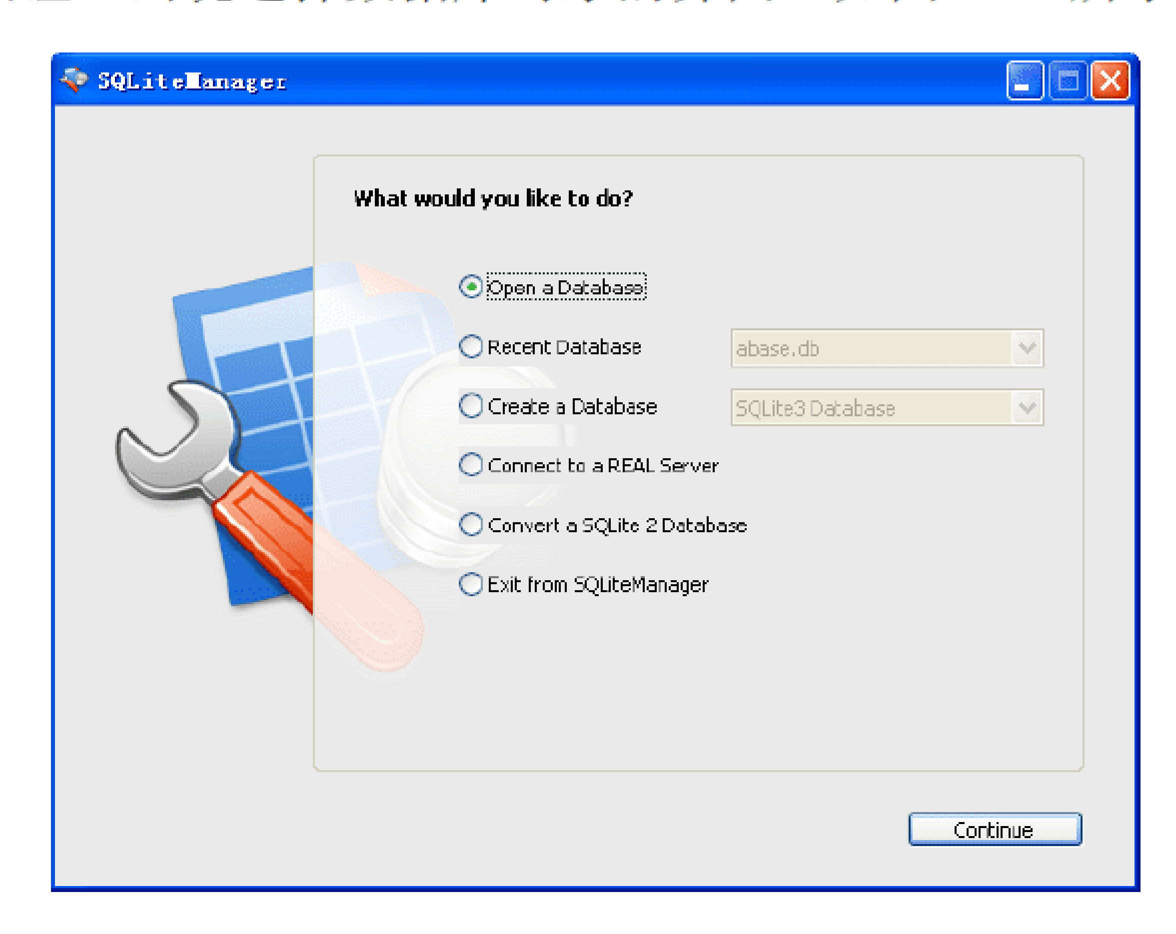


图 10-2　选择数据库类型

选择 Open a Database 单选按钮，打开一个数据库，或者选中 Create a Database。创建一个数据库。这里我们选择创建一个数据库，名称为 userDB，保存类型为 DB(*.db)，如图 10-3 所示。

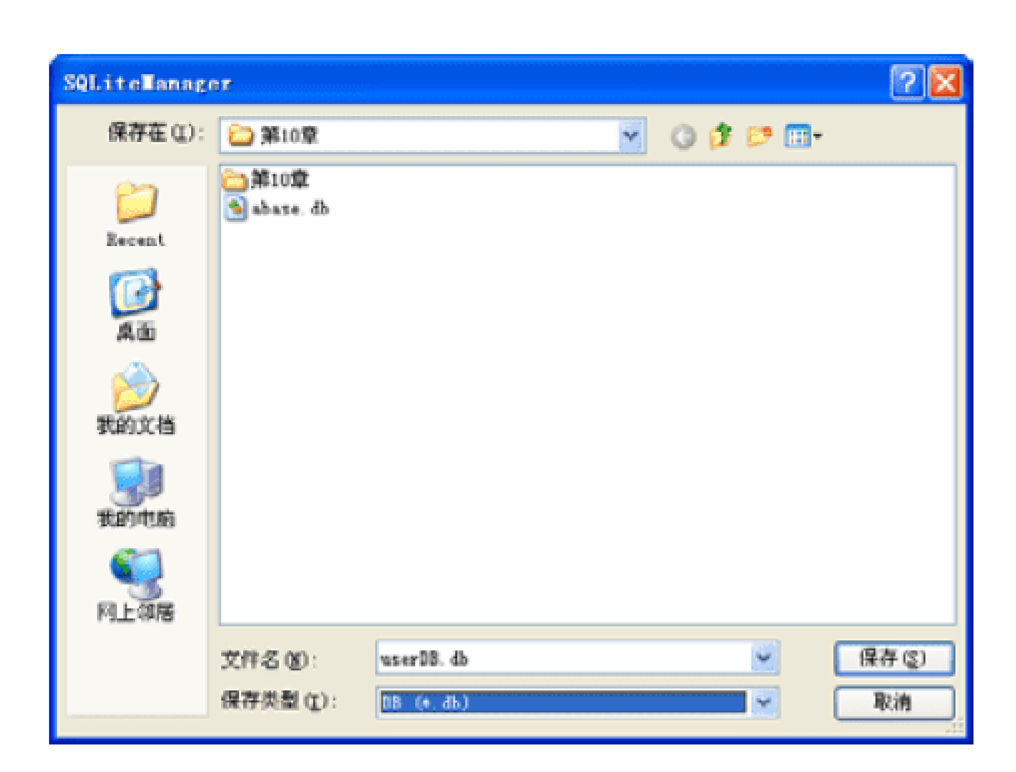


图 10-3　创建一个数据库

保存之后，即打开数据库编辑器，如图 10-4 所示。

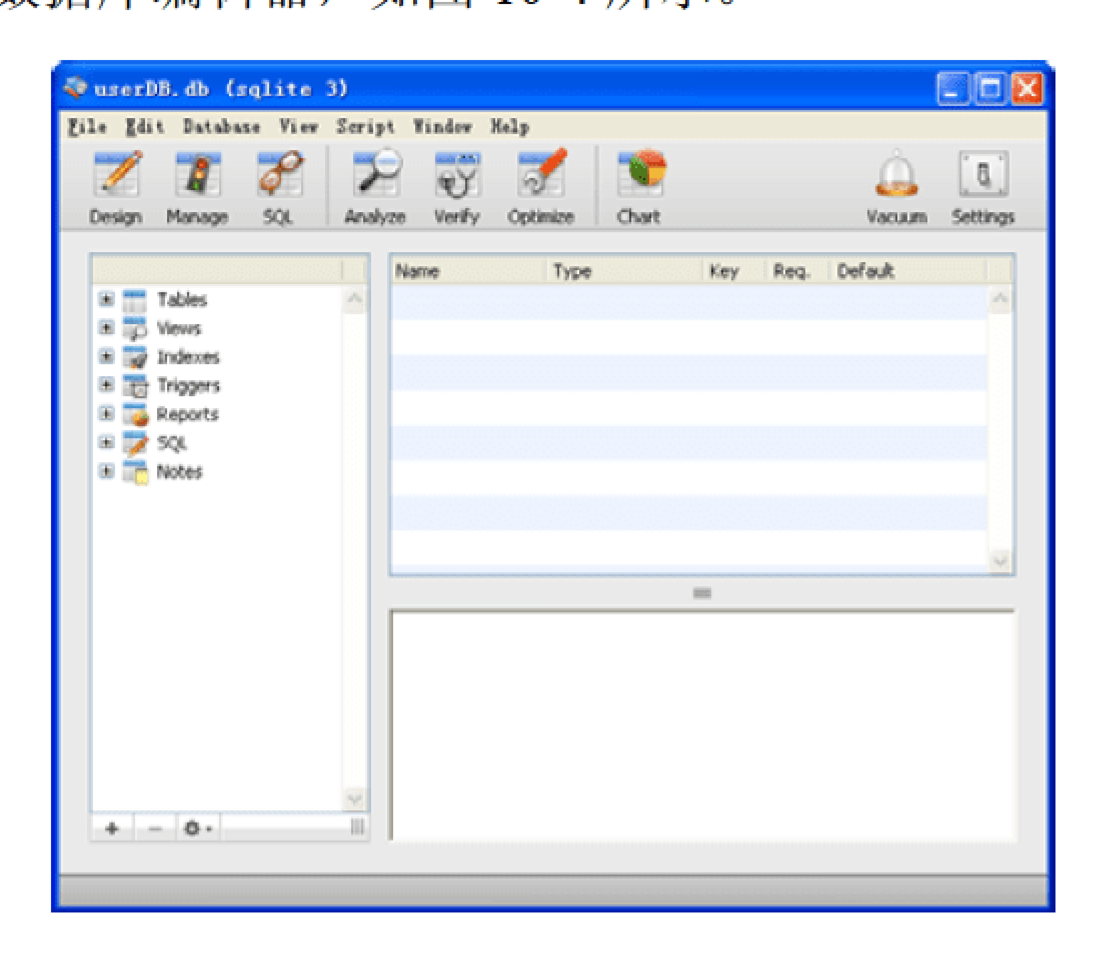


图 10-4　维护数据库

由于在 Python 2.5 中已经自带了 sqlite3 模块，所以要使用 SQLite 数据库，只需要导入 sqlite3 模块即可。

在正式接触连接 SQLite 数据库之前，先来了解一下有关操作数据库的基本对象。首先要介绍的是数据库连接对象，代码如下：

```
conn=sqlite3.connect('userDB.db')
```

上述代码中的 conn 就是一个数据库连接对象。它有几种操作，如表 10-5 所示。

表 10-5　数据库连接对象的操作

操　作	说　明
commit()	提交事务
rollback()	事务回滚

续表

操　作	说　明
close()	关闭一个数据库连接
cursor()	创建一个游标

数据库连接成功之后，接着就要对 SQL 语句进行操作了。在这里所有 SQL 语句的执行都要在游标对象下进行。创建游标对象的代码如下：

```
cur=conn.cursor()
```

游标对象 cur 可以执行的操作如表 10-6 所示。

表 10-6　游标对象的一些操作

操　作	说　明
execute()	执行一条 SQL 语句
executemany()	执行多条 SQL 语句
close()	关闭游标
fetchone()	从结果中取出一条记录
fetchmany()	从结果中取出多条记录
fetchall()	从结果中取出所有记录
scroll()	游标滚动

接下来看一下连接 SQLite 数据库的步骤，大致可以分为以下 6 个步骤。

(1) 导入 sqlite3 模块。

(2) 调用 connect()创建数据库连接，返回对象 conn。

(3) 调用 conn.execute()方法创建表结构或者插入数据。如果设置了手动提交，则需要调用 conn.commit()方法提交插入的数据。

(4) 调用 conn.cursor()方法返回游标，然后通过 cur.execute()方法查询数据库。

(5) 调用 cur.fetchall()、cur.fetchmany()或者 cur.fetchone()方法返回查询结果。

(6) 关闭游标对象 cur 和数据库连接对象 conn。

接下来，按照连接 SQLite 数据库步骤来做一个小例子。前面我们已经创建了一个名称为 userDB 的数据库，下面就以 userDB 为例，使用 sqlite3 的 connect()方法连接到该数据库，从而对数据进行增、删、改、查等操作，其代码如下：

```
import sqlite3                          #导入 sqlite3 模块
conn=sqlite3.connect('userDB.db')       #连接数据库 userDB，如果 userDB 不存在，则
创建数据库 userDB
conn.execute("create table if not exists address(id integer primary key
autoincrement,name varchar(128),address varchar(128))")        #如果 address
表不存在，则创建表 address
conn.execute("insert into address(name,address)values('dcy','zhengzhou')")
                                        #添加一条数据到表 address
conn.execute("insert into address(name,address)values
('duanchunyang','beijingchaoyangqu')") #向表 address 中再添加一条数据
```

```
conn.commit()                          #手动提交
cur=conn.cursor()                      #返回一个游标对象 cur，该对象主要用于查询数据
cur.execute("select * from address")   #调用游标对象 cur 的 execute 方法查询表
address 中的数据
res=cur.fetchall()                     #调用游标对象 cur 的 fetchall()方法返回表
address 中的所有数据
print "address:",res                   #输出结果集
for line in res:                       #输出结果集
    for f in line:
        print f
cur.close()                            #关闭游标对象 cur
conn.close()                           #关闭数据库链接对象 conn
```

运行程序，执行结果如下：

```
address: [(1, u'dcy', u'zhengzhou'), (2, u'duanchunyang',
u'beijignchaoyangqu'), (3, u'dcy', u'zhengzhou'), (4, u'duanchunyang',
u'beijignchaoyangqu')
1
dcy
zhengzhou
2
duanchunyang
beijignchaoyangqu
3
dcy
zhengzhou
4
duanchunyang
beijignchaoyangqu
>>>
```

10.4.2　实例描述

我喜欢编程，并且很自信地认为只要学好 Java 和.NET 这两种语言，就可以闯天下了。从事编程以来，经常听到的一句话就是：编程语言都是相通的，只不过是语法不同而已。这句话慢慢成为我征服其他语言的动力，Python 语言就是一个很好的例子。从我开始接触嵌入式 SQLite 数据库的欣喜，接着头昏脑胀，最后感到欣慰。在这个漫长的过程中，我深刻体会到：如果你对一门语言的语法一知半解，那么你永远只能是别人手下的一个员工。下面就是我针对连接 SQLite 数据库的基本语法做的一些功能，请看。

10.4.3　实例应用

【例 10-2】SQLite 数据库相关操作。

(1) 创建一个名为 Jiben.py 的文件。

(2) 在 Jiben.py 文件中添加代码。

```
import sqlite3                          #导入 sqlite3 模块
conn=sqlite3.connect('userDB.db')       #连接数据库 userDB，如果 userDB 不存在，则创建数
据库 userDB
cur=conn.cursor()
print '----------未处理之前的数据-----------'
cur.execute("select * from address")
res=cur.fetchall()                  #调用游标对象 cur 的 fetchall()方法，返回表 address
中的所有数据
print "address:",res                #输出结果集

for line in res:                    #输出结果集
    for f in line:
        print f

strUpdate=raw_input('请选择您要修改某条数据的编号：')
strId=raw_input('请选择您要删除某条数据的编号：')

conn.execute("update address set name='maxianglin' where id="+strUpdate)
conn.execute("create table if not exists address(id integer primary key
autoincrement,name varchar(128),address varchar(128))")        #如果 address
表不存在，则创建表 address
conn.execute("insert into address(name,address)values('dcy','zhengzhou')")
    #添加一条数据记录到表 address 中
conn.execute("delete from address where id="+strId)
conn.commit()                       #手动提交
print '-------------处理之后的数据----------------'
cur.execute("select * from address")    #调用游标对象 cur 的 execute 方法，查询表
address 中的数据
res=cur.fetchall()                  #调用游标对象 cur 的 fetchall()方法，返回表 address
中的所有数据
print "address:",res                #输出结果集

for line in res:                    #输出结果集
    for f in line:
        print f
cur.close()                         #关闭游标对象 cur
conn.close()                        #关闭数据库连接对象 conn
```

(3) 保存修改好的代码。

10.4.4　运行结果

运行程序，执行结果如下：

```
>>>
---------未处理之前的数据-----------
address: [(2, u'duanchunyang', u'beijignchaoyangqu'), (32, u'dcy',
u'zhengzhou'), (34, u'dcy', u'zhengzhou')]
2
duanchunyang
beijignchaoyangqu
32
dcy
zhengzhou
34
dcy
zhengzhou
请选择您要修改某条数据的编号：
```

当输入待修改的某条数据的编号为 32，或者输入待删除的某条数据的编号为 34 时，执行结果如下：

```
请选择您要修改某条数据的编号：32
请选择您要删除某条数据的编号：34
-------------处理之后的数据----------------
address: [(2, u'duanchunyang', u'beijignchaoyangqu'), (32, u'maxianglin',
u'zhengzhou'), (35, u'dcy', u'zhengzhou')]
2
duanchunyang
beijignchaoyangqu
32
maxianglin
zhengzhou
35
dcy
zhengzhou
>>>
```

10.4.5 实例分析



源码解析

在本实例中，使用了 connect()方法返回一个数据库连接对象 conn，接着创建一个游标对象 cur，并使用该游标对象 cur 的 fetchal()方法将表中的数据全部读取并显示出来。之后根据输入的编号，使用连接对象 conn 的 execute()方法分别进行相关操作。最后分别使用游标对象 cur 和连接对象 conn 的 close()方法，将对象关闭。

10.5　常见问题解答

10.5.1　持久化模块anydbm问题



持久化模块 anydbm 问题。
网络课堂：http://bbs.itzcn.com/thread-13730-1-1.html

我对持久化很感兴趣，因此使用 anydbm 模块写了一段代码。

```
import anydbm
def createData ():
    try:
        db=anydbm.open('db.dat','c')
        db['int']=1
        db['float']=2.0
        db['string']='I Love Python'
        db['key']='value'
    finally:
        db.close()

def loadData ():
    db=anydbm.open('db.dat','r')
    for item in db.items() :
        print item
        db.close()

if __name__=='__main__':
    createData()
    loadData()
```

运行程序，执行结果却引发了下面的异常。

```
Traceback (most recent call last):
  File "ce.py", line 19, in <module>
    createData()
  File "ce.py", line 5, in createData
    db['int']=1
  File "D:\Program Files\Python\lib\bsddb\__init__.py", line 230, in
__setitem__
    _DeadlockWrap(wrapF)  # self.db[key] = value
  File "D:\Program Files\Python\lib\bsddb\dbutils.py", line 62, in DeadlockWrap
    return function(*_args, **_kwargs)
  File "D:\Program Files\Python\lib\bsddb\__init__.py", line 229, in wrapF
    self.db[key] = value
TypeError: Data values must be of type string or None.
```

真是越着急，越看不出来哪里出错了。希望高手指点一二，非常感谢！

【解决办法】很高兴为你解答。

首先需要明白的是：anydbm 和 dbhash 模块的使用非常相似，存入数据库对象中的键和值都必须为字符串，否则就会引发 TypeError 异常。如果你将 createData()方法中的代码做如下修改。

```
def createData ():
    try:
        db=anydbm.open('db.dat','c')
        db['int']='1'
        db['float']='2.0'
        db['string']='I Love Python'
        db['key']='value'
    finally:
        db.close()
```

再次执行程序，显示结果如下：

```
('float', '2.0')
('string', 'I Love Python')
('key', 'value')
('int', '1')
```

一切正常！你明白了吧。

10.5.2 持久化模块shelve问题



持久化模块 shelve 问题。

网络课堂：http://bbs.itzcn.com/thread-13730-1-1.html

我知道模块 anydbm 和 dbhash 存储的值只能为字符串，而不能是数字、字典或者元组，而模块 shelve 恰能弥补这些缺陷。于是我很想写一段代码来证明字典、元组或者数字如何存储在 shelve 模块对象中。苦于水平一般，能力有限，希望好心者给一段代码并指点一二，在下感激不尽！

【解决办法】很高兴为你解答。

这是我写的一段代码，只是不知道能不能满足你的要求。代码如下：

```
import shelve
def myLove ():
    db=shelve.open('mydb','r')
    db[1]=[5,17,23,58]
    db[2]=158962
    db[3]={'dream':'梦想','beautiful':'美好'}
    db[4] = "一切是如此的不可思议，如果不是尝过苦涩的滋味，我想我不会发现，你早已悄悄的
在我心中，注入了一口甜蜜，这是触动的感觉"
    db.close()
def myShow ():
    db=shelve.open('mydb','r')
    for item,value in db.items() :
        print item,value
        db.close()

if __name__=='__main__':
    myShow()
```

运行程序，执行结果如下：

```
>>>
2 ['ltt', '15093077824', 'beijing', 'myworld', 2500]
```

```
4 ['hoppy', '15093077826', 'shenzhen', 'myworld', 3500]
dcy ['dcy', '15093077823', 'shanghai', 'myworld', 4000]
ltt ['ltt', '15093077824', 'beijing', 'myworld', 2500]
hoppy ['hoppy', '15093077826', 'shenzhen', 'myworld', 3500]
mykeyvalue {'food': 'spam', 'taste': 'yum'}
宝珠 一切是如此的不可思议，如果不是尝过苦涩的滋味，我想我不会发现，你早已悄悄的在我心中，注
入了一口甜蜜，这是触动的感觉
1 ['dcy', '15093077823', 'shanghai', 'myworld', 4000]
3 ['mxl', '15093077825', 'tianjin', 'myworld', 2000]
mxl ['mxl', '15093077825', 'tianjin', 'myworld', 2000]
secretCombination [5, 17, 23, 58]
account 158962
>>>
```

这个例子想必你能明白吧。

10.5.3　Python中数据库连接问题



Python 中的数据库连接问题。
网络课堂：http://bbs.itzcn.com/thread-13730-1-1.html

我使用的工具是 Python 2.5，由于 Python 2.5 中自带有 sqlite3 库，当使用嵌入式数据库 SQLite 时，只需直接导入 sqlite3 即可，这对我来说不难。但要是连接 SQL Server、MySQL 或者 Oracle 数据库的话，还真不容易。最近我在网上找了好多资料，看得我迷迷糊糊，按照步骤一步一步地安装，到最后还是出现错误。哎，真是头疼！在这里我恳请大侠给一个可靠一点的连接 SQL Server 数据库的代码，包括下载的包或者需要安装的 exe 文件，感激不尽！

【解决办法】很高兴为你解答。

你上面说道，你安装 Python 语言的版本是 Python 2.5，如果你需要连接 SQL Server 数据库，那么你需要下载 pymssql-0.8.0.win32-py2.5.exe，下载的地址是 http://d.download.csdn.net/down/214710/wuyoubf。之后单击 pymssql-0.8.0.win32-py2.5.exe 进行安装，如图 10-5 所示。

单击“下一步”按钮，然后选中 Python Version2.5(found inreqistry)，其中名称为 Python Directory 的文本框的目录为 D:\Program Files\Python\，名称为 Installation Directory 的文本框目录为 D:\Programs Files\Python\Lib\site-packages\。需要注意的是，我们将 Python 2.5 安装到 D:\Programs Files\目录下，如图 10-6 所示。

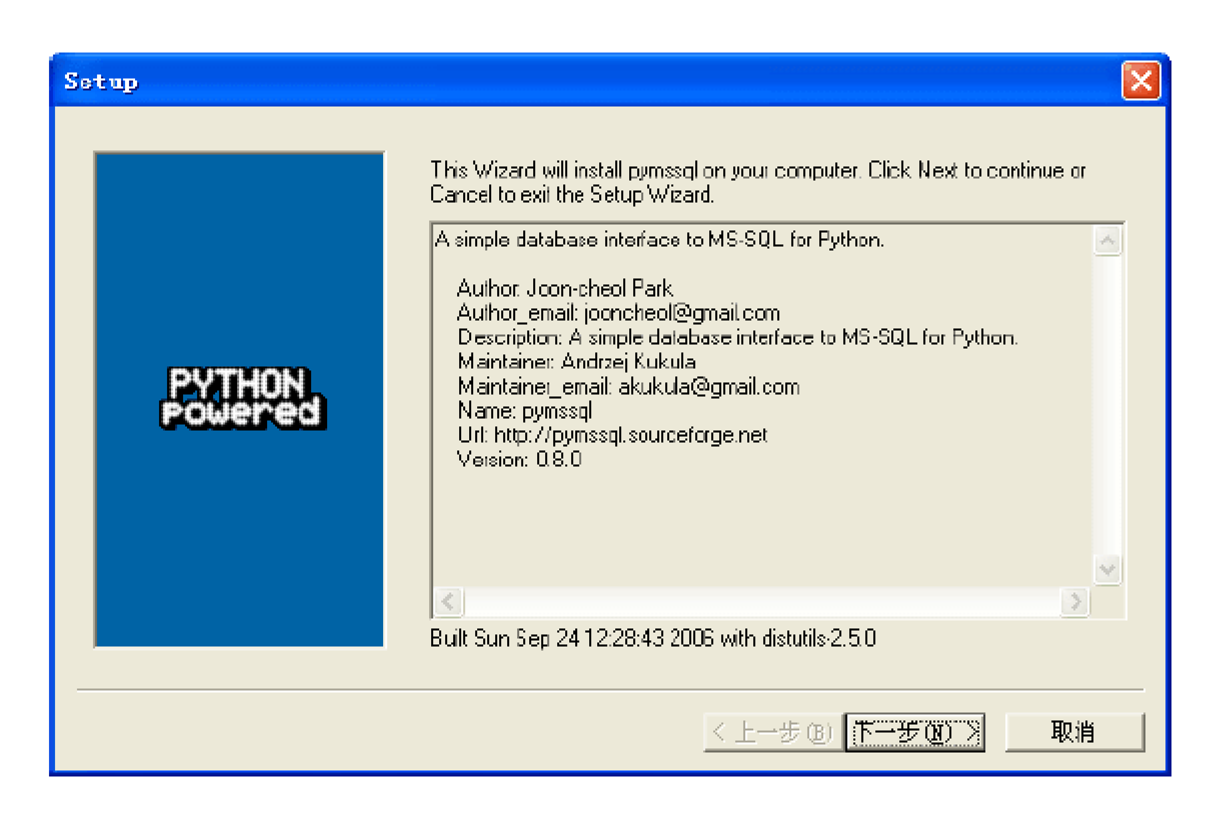


图 10-5　单击.exe文件后显示的效果

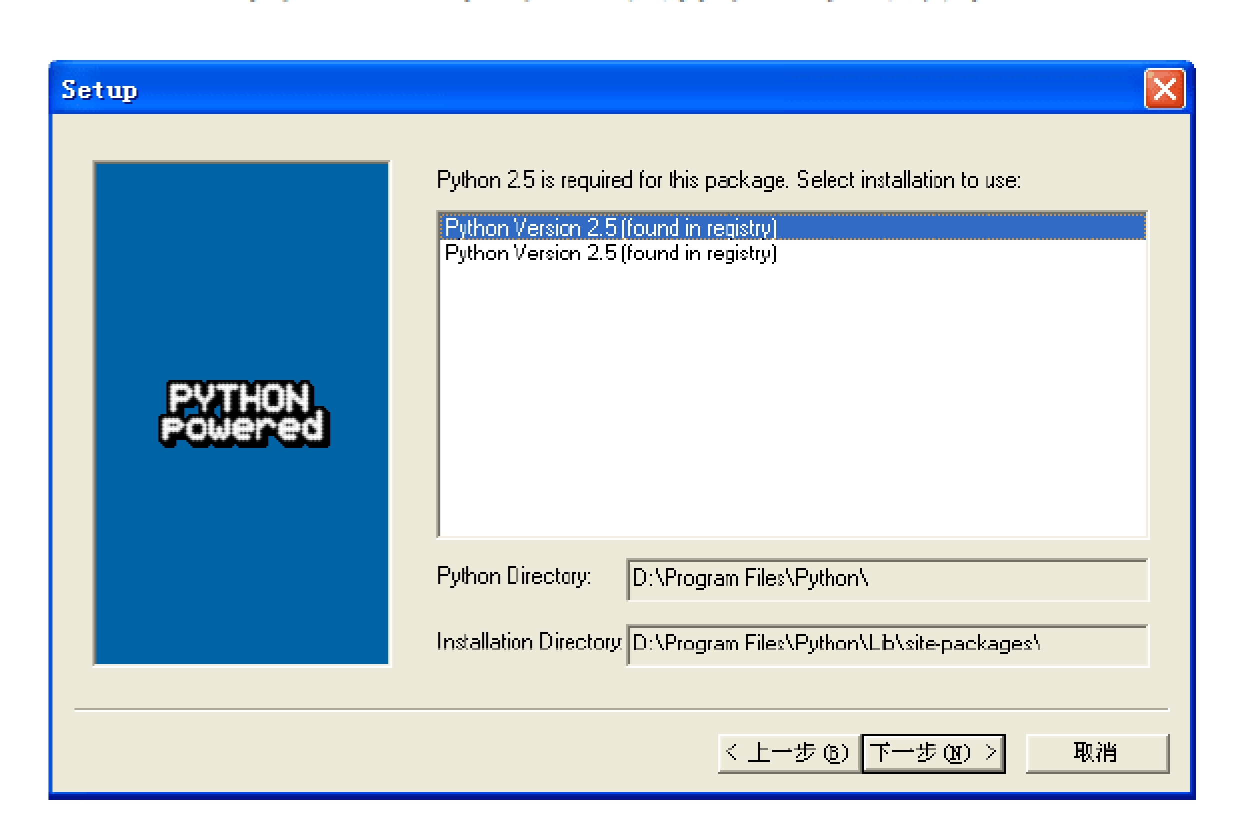


图 10-6　单击按钮“下一步”之后显示的效果

接着一直单击“下一步”按钮，直到单击“完成”按钮之后，才算安装成功。

下面来看一下我写的一段连接 SQL Server 数据库的代码。

```
import pymssql
conn=pymssql.connect(host='.\SQLEXPRESS',user='sa',password='sa',database='
UserDB')
cur=conn.cursor()
cur.execute('select * from users')
res=cur.fetchall()
for a in res:
    for b in a:
        print b
```

保存修改好的代码，运行程序，执行结果如下：

```
>>>
1
dcy
dcy
luzhai
15093077823
2
dcy
dcy
luzhai
15093077823
```

```
3
dcy
dcy
luzhai
15093077823
>>>
```

这样，你明白了吧。

10.6　习　　题

一、填空题

(1) 在 Python 中，如果想要实现一个持久化操作，可以使用模块 dbhash、anydbm 和__________。

(2) 有这样一段代码：

```
import __________
db=shelve.open('mydb')
db['1']=['dcy','15093077823','shanghai','myworld',4000]
db['2']=['ltt','15093077824','beijing','myworld',2500]
db['3']=['mxl','15093077825','tianjin','myworld',2000]
print db
db.close()
```

如果上述代码能准确无误地执行，那么在空白处需要导入的模块是__________。

(3) 在 Python 语言中进行数据库连接时，我们使用__________方法，可以返回一个连接对象。

(4) 假如我们有一个游标对象 cur，那么可以使用游标对象的__________方法来查询一条 SQL 语句。

(5) 有这样一段代码：

```
import sqlite3
conn=sqlite3.connect('MyGood.db')
conn.execute("create table if not exists address(id integer primary key
autoincrement,name varchar(128),address varchar(128))")
conn.execute("insert into
address(name,address)values('duanchunyang','beijignchaoyangqu')")
conn.commit()
cur=conn.cursor()
cur.execute("select * from address")
res=cur. __________
print "address:",res
cur.close()
conn.close()
```

如果我想获得结果集中的所有数据，那么在空白处需要填写的方法是__________。

二、选择题

(1) 在下面的几个选项中，使用模块进行持久化操作的代码，正确的是__________。

A.

```
import shelve
db=shelve.open('mydb')
db['1']=['myworld',4000]
db['2']=['myworld',2500]
db['3']=['myworld',2000]
print db
db.close()
```

B.

```
import anydbm
db=anydbm.open('mydb')
db['1']=['myworld',4000]
db['2']=['myworld',2500]
db['3']=['myworld',2000]
print db
db.close()
```

C.

```
import dbhash
db=dbhash.open('mydb')
db['1']=['myworld',4000]
db['2']=['myworld',2500]
db['3']=['myworld',2000]
print db
db.close()
```

D.

```
import dbhash
db= anydbm.open('mydb')
db['1']=['myworld',4000]
db['2']=['myworld',2500]
db['3']=['myworld',2000]
print db
db.close()
```

(2) 如果我们有这样一段代码：使用 connect()方法返回一个连接对象 conn，接着使用连接对象 conn 的 cursor()方法返回一个游标对象 cur，那么使用 cur 对象__________方法才能获得结果集中的所有行。

A. fetchmany()　　　　B. fetchone()

C. fetchall()　　　　D. executemany()

(3) 有这样一段代码：

```
import sqlite3
conn=sqlite3.connect('okDB.db')
conn.execute("create table if not exists address(id integer primary key
autoincrement,question varchar(128),answer varchar(128))")
conn.execute("insert into address(question,answer)values('do you
know','People fell from 1 floor fell from 10th floor and have what
distinction' )")
conn.execute("insert into address(question,answer)values('Do you like
person','If not, you will not have the qualifications commentary I ')")
cur=conn.cursor()
cur.execute("select * from address")
res=cur.fetchmany(2)
res1=cur.fetchone()
```

```
print "question:",res1
for line in res:
    print line
cur.close()
conn.close()
```

运行程序，执行的结果正确的是__________。

A.

```
>>>
question: (3, u'do you know', u'People fell from 1 floor fell from 10th floor
and have what distinction')
(1, u'do you know', u'People')
>>>
```

B.

```
>>>
question: (3, u'do you know', u'People fell from 1 floor fell from 10th floor
and have what distinction')
(2, u'Do you like person', u'no')
>>>
```

C.

```
>>>
question: (3, u'do you know', u'People fell from 1 floor fell from 10th floor
and have what distinction')
>>>
```

D.

```
>>>
question: (3, u'do you know', u'People fell from 1 floor fell from 10th floor
and have what distinction')
(1, u'do you know', u'People')
(2, u'Do you like person', u'no')
>>>
```

三、上机练习

上机练习： 实现对数据库的增、删、改、查等功能。

本章主要介绍了如何对数据进行持久化操作，通过模块 dbhash 或者 shelve 都可以将数据进行持久化存储。将数据写入数据库可以永久保存，难道不是持久化操作的一种？本章接着介绍了数据库的连接对象和游标对象的方法，进而以嵌入式数据库 SQLite 为例对数据进行操作。

本次上机练习就是对数据库中的数据实现增、删、改、查等操作，最后提交到数据库以达到永久保存的目的。

当您选择“添加”字符时，执行结果如下：

```
>>>
请选择您的下一步：
(添加) 往数据库中添加内容    (删除) 删除数据库中的内容    (修改)修改数据的内容
(查询) 查询数据的内容
选择您想要进行的操作:添加
------------欢迎您使用添加数据的功能-------------
请输入您的用户名: world
```

```
请输入您的密码：world
请输入您的地址：zhengzhou
请输入您的联系电话：15093077823
--------恭喜你，添加成功-------
----------添加之后的数据-----
(21, 'tian', 'tian', 'tian', 'tian')
(19, 'my', 'my', 'my', 'my')
(20, 'wo', 'wo', 'wo', 'wo')
(23, 'world', 'world', 'zhengzhou', '15093077823')
(11, 'dcy', 'dcy', 'dcy', 'dcy')
(12, 'dcy', 'dcy', 'dcy', 'cyd')
>>>
```

当你输入“删除”字符时，执行结果如下：

```
>>>
选择您想要进行的操作:删除
请输入您想要删除用户的编号：20
------------恭喜你，删除成功------------
-----------删除之后所显示的数据--------------
(21, 'tian', 'tian', 'tian', 'tian')
(19, 'my', 'my', 'my', 'my')
(23, 'world', 'world', 'zhengzhou', '15093077823')
(11, 'dcy', 'dcy', 'dcy', 'dcy')
(12, 'dcy', 'dcy', 'dcy', 'cyd')
>>>
```

当你输入“修改”字符时，执行结果如下：

```
>>>
选择您想要进行的操作:修改
请输入您想要修改用户的编号：11
请输入您修改的用户名：niaoyu
请输入您修改的密码：huaxiang
请输入您修改的地址：zhengzhou
请输入您修改的联系电话：15093077823
-----------更新之后的数据显示-------------
(21, 'tian', 'tian', 'tian', 'tian')
(19, 'my', 'my', 'my', 'my')
(23, 'world', 'world', 'zhengzhou', '15093077823')
(11, 'niaoyu', 'huaxiang', 'zhengzhou', '15093077823')
(12, 'dcy', 'dcy', 'dcy', 'cyd')
>>>
```

当你输入“查询”字符时，执行结果如下：

```
>>>
选择您想要进行的操作:查询
请输入您想要查询用户的编号：11
------------恭喜你，查询的数据如下所示------------
您查询的数据编号为：11
您查询的数据名称为：niaoyu
您查询的数据密码为：huaxiang
您查询的数据地址为：zhengzhou
您查询的数据电话为：15093077823
>>>
```



第 11 章　让信息自由联通——Python网络功能

内容摘要

Python 是一个很强大的网络编程工具，具有很多针对常见网络协议的库，在库的顶部可以获得抽象层，这样就可以集中精力在程序的逻辑处理上，而不是停留在网络实现的细节上。如何有效地构建具有丰富功能的客户端和服务器端一直是使用 Python 进行网络编程的目标。

本章将详细介绍 Python 网络编程，包括网络设计模块、服务器端和客户端之间的通信、异步通信方式和 Twisted 网络框架等知识。

学习目标

- 了解 TCT/IP 网络互联模型。
- 掌握 Socket 基础知识。
- 掌握服务器端和客户端之间的通信技术。
- 掌握异步通信方式。
- 了解 Twisted 网络框架的使用。

11.1　网络模型介绍

随着网络技术的迅速发展，网络应用也变得越来越丰富。在介绍网络应用之前，先来了解一下 Python 中的网络模型都有哪些。本节将详细介绍 Python 中的网络模型：开放式系统互联参考模型和 TCP/IP 模型。

视频教学：光盘/videos/11/网络模型介绍.avi　　长度：9 分钟

11.1.1　基础知识——OSI简介

OSI(Open System Interconnect，开放式系统互联)，是由国际标准化组织(International Organization for Standardization，ISO)制定的。网络发展中一个重要里程碑便是 ISO 对 OSI 七层网络模型的定义，它不但成为以前的和后续的各种网络技术评判、分析的依据，也成为网络协议设计和统一的参考模型。OSI 模型把网络通信的工作分为 7 层，分别是物理层、数据链路层、网络层、传输层、会话层、表示层和应用层。

1. 第一层：物理层(Physical Layer)

规定通信设备的机械、电气、功能和过程特性，用以建立、维护和拆除物理链路连接。物理层的主要功能是为数据端设备提供传送数据的通路，数据通路可以是一个物理媒体，也可以是多个物理媒体连接成完整的数据传输，包括激活物理连接，传送数据以及终止物理连接。所谓激活，就是不管有多少物理媒体参与，都要在通信的两个数据终端设备间连接起来，形成一条通路。

2. 第二层：数据链路层(DataLink Layer)

在物理层提供比特流服务的基础上，建立相邻节点之间的数据链路，通过差错控制提供数据帧(Frame)在信道上无差错的传输，并进行各电路上的动作系列。这个层次的数据单位称为帧。数据链路层包括两个重要的子层：逻辑链路控制层(Login Link Control，LLC)和介质访问控制层(Media Access Control，MAC)。LLC 用来对节点间的通信链路进行初始化，并防止链路中断，确保系统的可靠通信。而 MAC 则用来检测包含在数据帧中的地址信息。这里的地址是链路地址或物理地址，是在设备制造的时候设置的。网络上的两种设备不能有相同的物理地址，否则会造成网络信息传送失败。

3. 第三层：网络层(Network Layer)

定义数据的寻址和路由方式。这一层负责对子网之间的数据选择路由，并实现网络互联等功能。

4. 第四层：传输层(Transport Layer)

传输层为数据提供端到端的传输。这是比网络层更高的层次，是主机到主机的层次。传输层对上层的数据进行分段并进行端到端的传输。另外，还提供差错控制和流量控制机制。

5. 第五层：会话层(Session Layer)

会话层用来为通信双方指定通信方式，负责建立和拆除会话。另外，此层将会在数据中插入校验码来实现数据的同步。

6. 第六层：表示层(Presentation Layer)

表示层为不同的用户提供数据和信息的转换，同时还提供解压缩和加解密服务。这一层保证了两个主机的信息可以互相解析。

7. 第七层：应用层(Application Layer)

应用层为操作系统或网络应用程序提供访问网络服务的接口。

OSI 的 7 层模型的关系如图 11-1 所示。

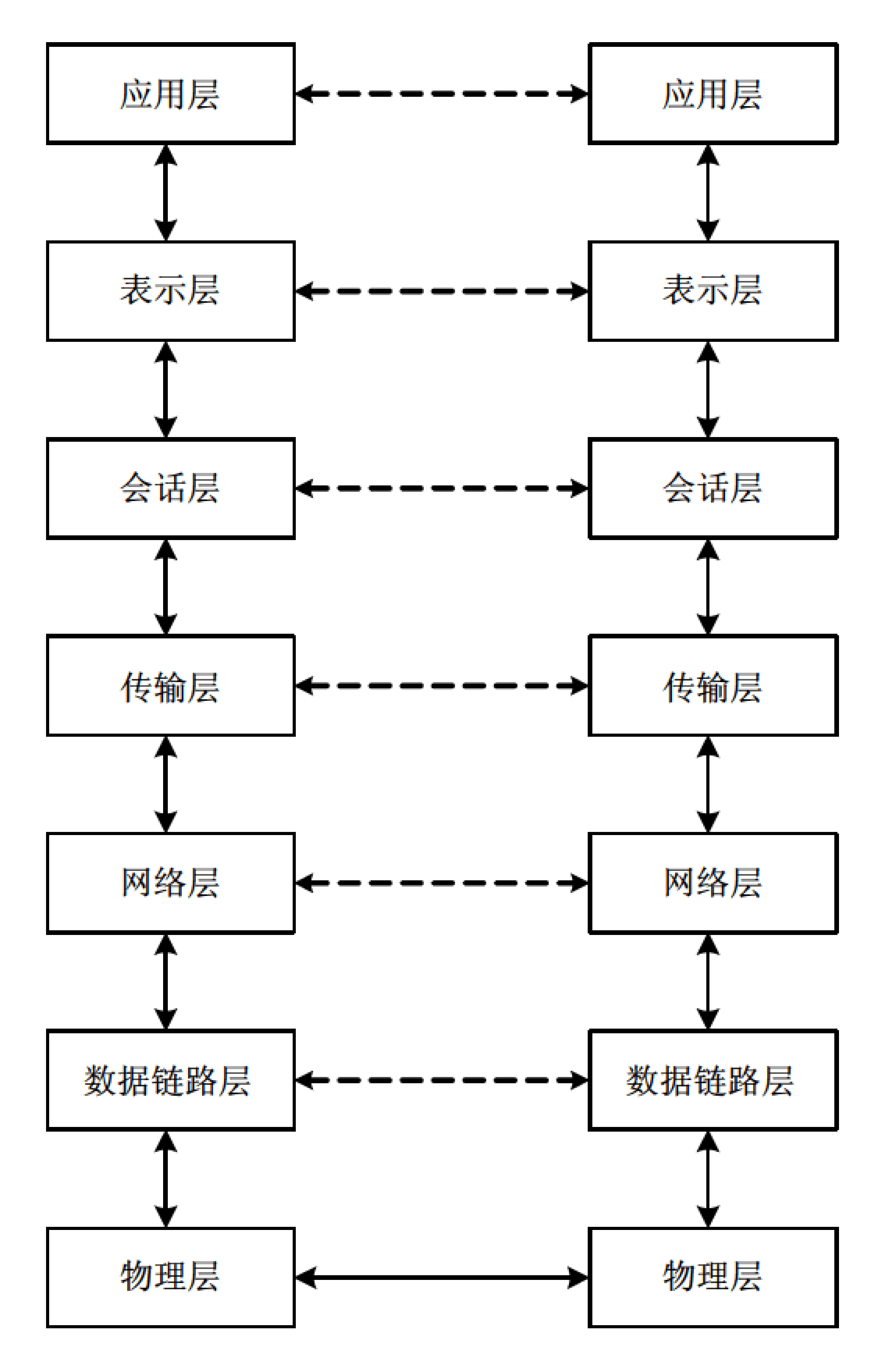


图 11-1　OSI模型图

在图 11-1 中，数据信息的实际流程用实线箭头标记，层次的关系用虚线箭头标记。在 OSI 的 7 层模型中，数据访问只会在上下相邻两层之间进行。

OSI 模型是一个通用的网络系统模型，并不是一个协议定义，所以实际上 OSI 模型从来没有被真正的实现过。但是，由于其模型的广泛指导性，现在的网络协议都已经纳入了 OSI 模型的范围之内。在其模型中，从下层至上层依次增加，其中物理层为第一层，数据链路层为第二层，依此类推。

11.1.2 基础知识——TCP/IP简介

TCP/IP(Transmission Control Protocol/Internet Protocol，传输控制协议/因特网互联协议，又叫网络通信协议)协议是 Internet 最基本的协议，是 Internet 国际互联网络的基础。简单地说，就是由网络层的 IP 协议和传输层的 TCP 协议组成的。TCP/IP 定义了电子设备(比如计算机)如何连入因特网，以及数据如何在它们之间传输的标准。

TCP/IP 是一个四层的分层体系结构。高层为传输控制协议，它负责收集信息或把文件拆分成更小的包。低层是网际协议，它处理每个包的地址部分，使这些包正确地到达目的地。由于 TCP/IP 协议的出现比 OSI 早，因此并不符合 OSI 模型，其对应关系如图 11-2 所示。

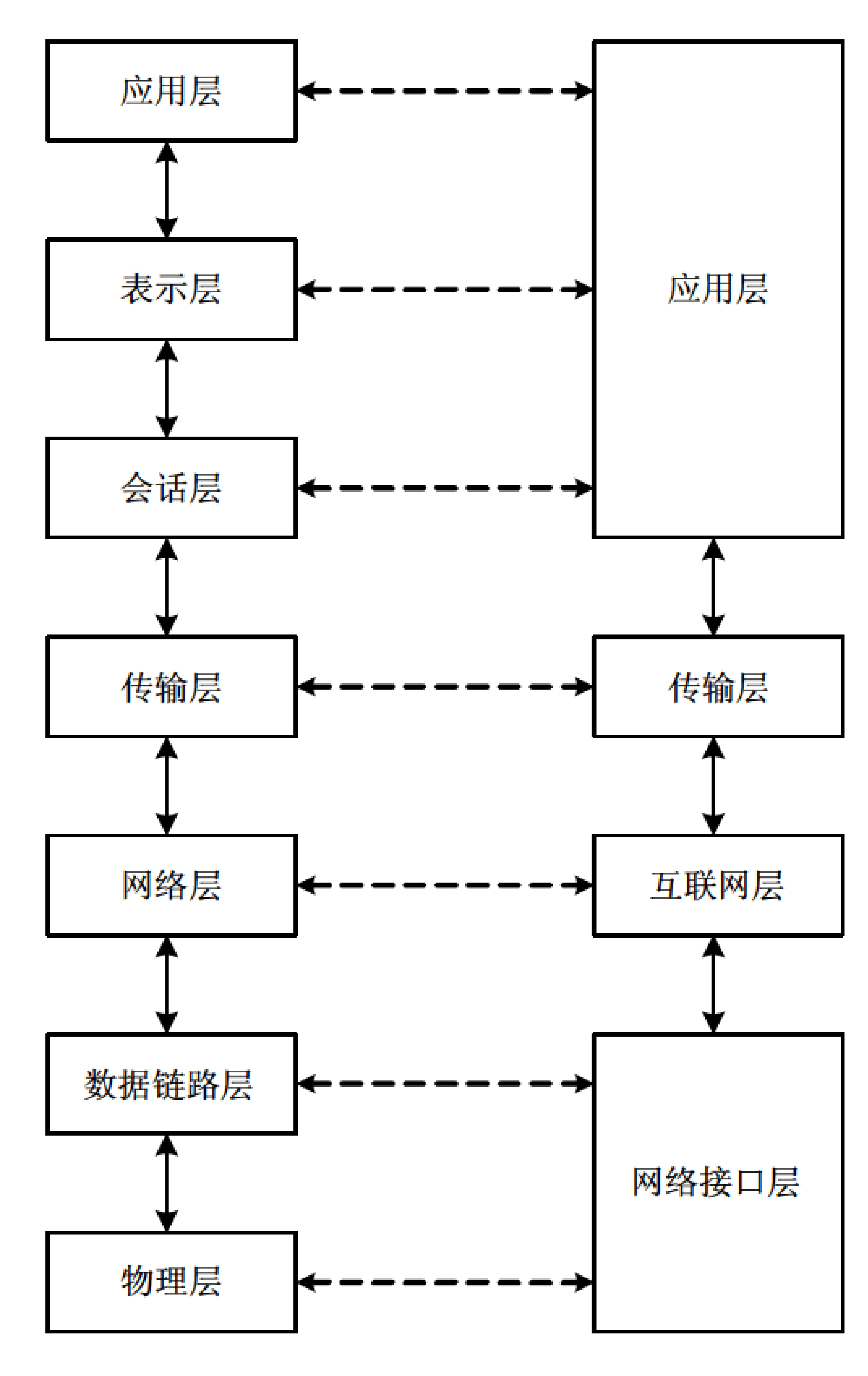


图 11-2 OSI模型和TCP/IP

在图 11-2 中，图的左边为 OSI 模型，右边为 TCP/IP 模型。可以看到，TCP/IP 模型并不关心互联网层以下的组成，而将数据输出统一成了网络接口层。这样，互联网层只需要将数据发往网络接口层就可以了，不需要关心下层的具体操作。在 OSI 模型中，则将这些功能分成了数据链路层和物理层，而且还进行了进一步的划分。在传输层和网络层大部分仍然一致，而对 OSI 中的上面三层，在 TCP/IP 模型中则将其合并为应用层。

在 TCP/IP 模型中，最主要的两个协议 TCP/IP 分别属于传输层和互联网层。在互联网层中，标识主机的方法是使用 IP 地址，例如 192.168.0.1 就是一个内网主机的 IP 地址。通过对 IP 地

址的类别划分，可以将整个 Internet 网络划分成不同的子网。而在传输层中，标识一个应用的方法是通过端口号来识别的，这些不同的端口号表示不同的应用。例如，80 端口一般是 HTTP 协议，23 端口号是 Telnet 协议等。在 TCP/IP 模型中，标识一个主机上的应用可以通过地址-端口号对来表示。

11.2　网络设计模块

在 Python 的标准库中有很多网络设计模块。除了明确处理网络事务的模块外，还有很多模块(比如用于在网络传输中处理各种形式的数据编码的模块)被认为是网络相关的。本节将详细的讨论常见的几个模块。

 **视频教学：**光盘/videos/11/Socket 模块.avi　　　　长度：14 分钟

11.2.1　基础知识——Socket模块

随着 TCP/IP 协议的流行，其中的 Socket 编程已经成为实现网络应用程序的基础。在本节中，主要讲解如何使用 Socket 模块。

1. Socket基础知识

在网络编程中，一个基本组件就是套接字(Socket)，套接字主要是两个程序之间的信息通道，程序可能分布在不同的计算机上，通过套接字相互发送信息。Python 大多数网络编程都隐藏了 Socket 模块的基本细节，并且不直接和套接字交互。随着 TCP/IP 协议的使用，套接字越来越多地被使用在网络应用程序的构建中。实际上，Socket 编程已经成为网络中传送和接收数据的首选方法。套接字最早是由伯克利在 BSD 中推出的一种进程间通信方案和网络互联的基本机制。现在，已经有多种相关的套接字实现，但是大部分仍然遵循最初的设计要求。

Socket 模块中的 socket()方法用于创建套接字，socket()方法的语法格式如下：

```
socket(socket_family, socket_type, protocol = 0)
```

其中：

- socket_family　该参数的值可以为 AF_UNIX 或 AF_INET。
- socket_type　该参数的值可以为 SOCK_STREAM 或 SOCK_DGRAM。
- protocol　该参数一般不赋值，默认值为 0。

套接字相当于应用程序访问下层网络服务的接口。使用套接字，使得用户可以在不同的主机之间进行通信，从而实现数据交换。图 11-3 是 Socket 使用图示。

一个正在被使用的套接字有着和其匹配的通信类型以及相关的进程信息，并且会和另外一个套接字交换数据。当前 Socket 规范支持两种类型的套接字，即流套接字和数据报套接字。其中，流套接字提供了双向有序且不重复的数据服务。而数据报套接字虽然支持双向的套接字，但是对报文的可靠性和有序性并不能保证。换句话说，通过数据报套接字接口获取的数据有可能是重复的，这需要上层的应用程序来进行区分。其实，还有更多的套接字类型，比如原始套接字类型等，这和系统中的实现相关。

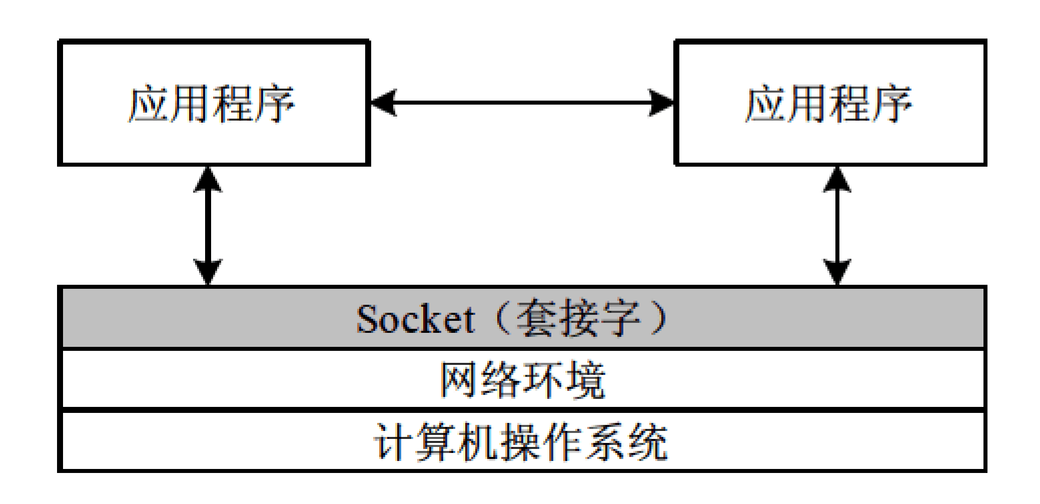


图 11-3　套接字的使用示意图

2. Socket的工作方式

套接字包括两个：服务器套接字和客户机套接字。套接字在工作的时候，将连接对端分为服务器端和客户端，这也是 C/S(Client/Server，客户端/服务器端)模式的由来。可以看出，套接字的对端通信是不对等的。根据不同的通信方式，通信协议可以分为对称协议和非对称协议。

处理客户端套接字通常比处理服务器端套接字容易，因为服务器必须准备随时处理客户端的连接，同时还要处理多个连接，而客户机只是简单地连接，完成事务、断开连接。一个套接字就是一个 Socket 模块中的 Socket 类的实例。它的实例化需要 3 个参数：

- 第一个参数是地址簇(默认是 socket.AF_INET)。
- 第二个参数是流(默认值为 socket.SOCK_STREAM)或数据报(默认值为 socket.SOCK_DGRAM)套接字。
- 第三个参数是使用的协议(默认为 0，使用默认值即可)。

不同的套接字类型有着不同的套接字地址。在 AF_UNIX 地址簇中使用一个简单的字符串；在 AF_INET 地址簇中使用的是 host 和 port 地址对，其中 host 为主机地址名、IP 地址或者 Internet 上的 URL，而 port 则是一个整数；在 AF_INET6 地址簇中用(host,port,flowinfo,scopeid)元组来表示，其中的 host 和 port 与 AF_INET 中相同。

客户端应用程序在生成套接字对象后，可以调用 bind()方法来绑定自己的请求套接字接口地址，然后调用 connect()方法来连接服务器端进程。当连接建立后，可以使用 send()和 recv()方法来传输数据。最后需要使用 close()方法将端口关闭。具体通信过程如图 11-4 所示。

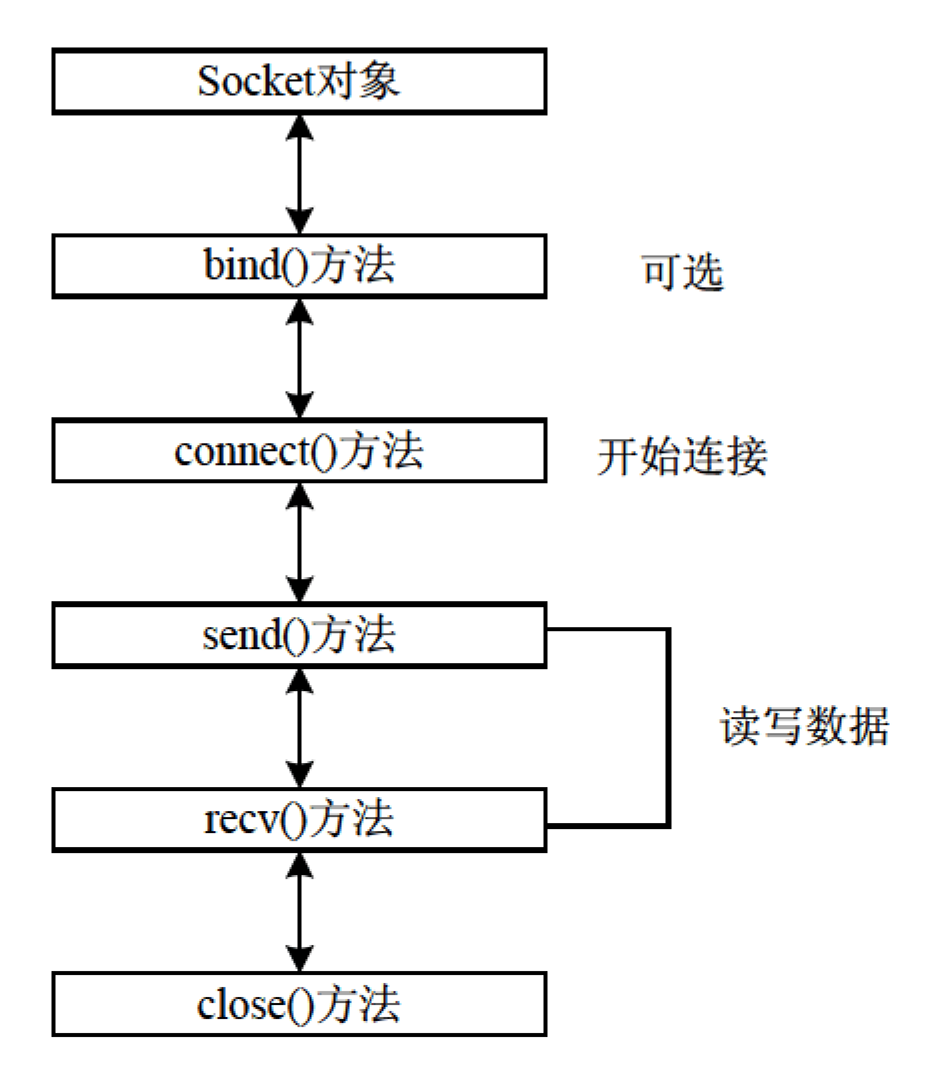


图 11-4　客户端的通信过程

服务器端套接字使用 bind()方法绑定一个套接字接口地址，接着使用 listen()方法监听客户端请求。当有客户端请求时，将通过 accept()方法来生成一个连接对象，然后通过此连接对象发送和接收数据。数据传输完毕，可以调用 close()方法将生成的连接关闭。服务器端通信过程的方法调用如图 11-5 所示。

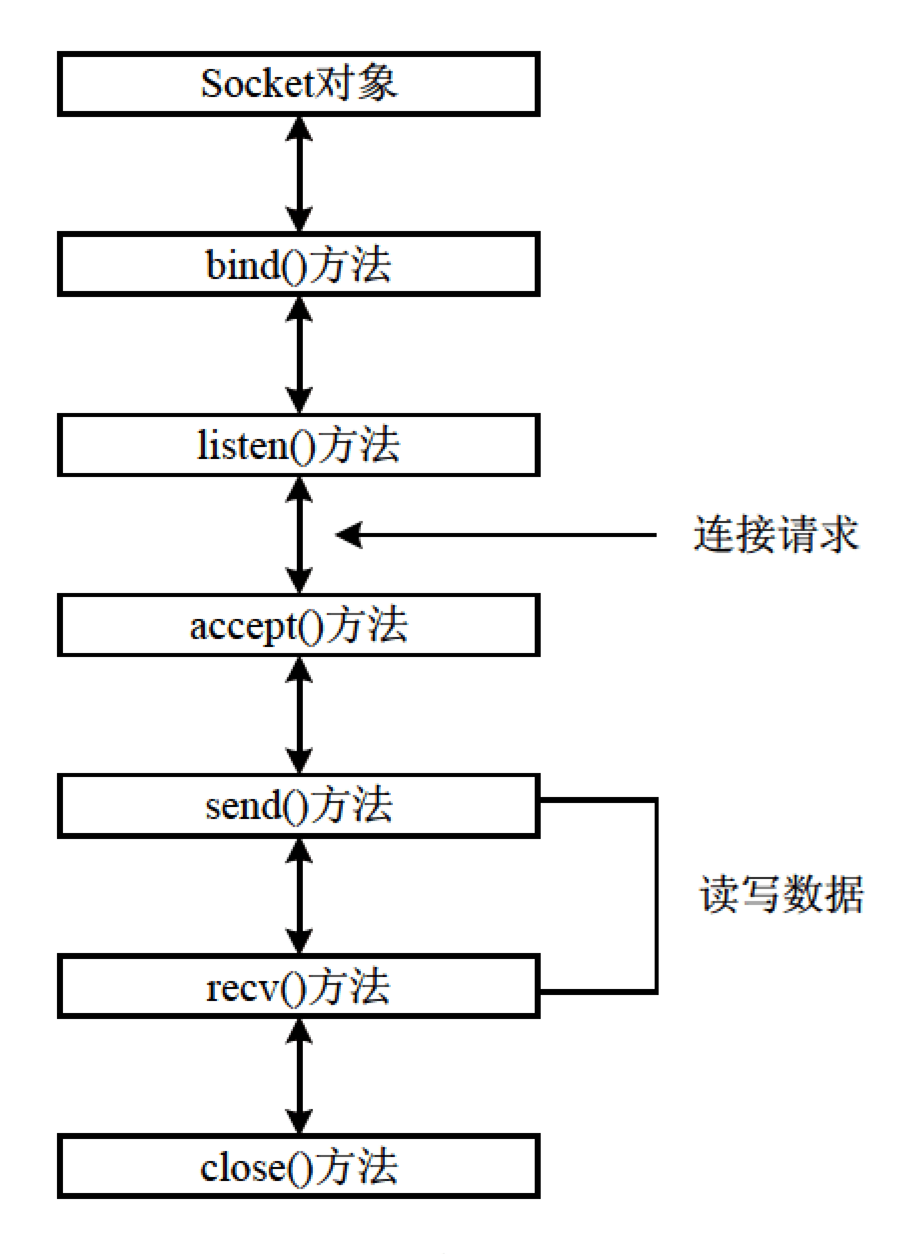


图 11-5　服务器端的通信过程

11.2.2　基础知识——urllib和urllib2 模块

在可使用的各种网络工作库中，功能最强大的是 urllib 和 urllib2。它们允许通过网络访问文件，就像这些文件存在于本地电脑上一样。只需通过一个简单的方法调用，几乎可以把任何 URL 所指向的文件用程序访问。

这两个模块的功能相似，但 urllib2 的功能更强大一些。如果只使用简单的下载，urllib 就足够了；如果需要使用 HTTP 验证或 Cookie 以及需要为协议写扩展程序，那么 urllib2 是不错的选择。

urllib2 是 Python 的一个获取 URL(Uniform Resource Locators，统一资源定址器)的模块。它用 urlopen()方法提供了一个非常简洁的接口，使得用各种各样的协议获取 URL 成为可能。同时提供了一个稍微复杂的接口来处理常见的状况(比如基本的认证、Cookies 和代理等)，这些都是由 opener 和 handler 的对象来处理的。

1. 打开远程文件

可以像打开本地文件一样打开远程文件，不同之处是可以使用只读模式，使用的是来自 urllib2 模块中的 urlopen()方法，该方法的语法格式如下：

```
urlopen(url, data=None, proxies=None)
```

该方法的参数说明如表 11-1 所示。

表 11-1　urlopen()方法的参数说明

参数名称	描　述
url	符合 URL 规范的字符串
data	向指定 URL 发送的数据字符串，GET 和 POST 都可以，但必须符合标准格式，格式为：key=value
proxies	代理服务器地址字典，如果未指定，在 Windows 平台上则依据 IE 的设置。不支持需要验证的代理服务器，例如 proxies={'http', 'http://www.someproxy.com:3128'}

下面创建一个实例，具体介绍如何使用 urlopen()方法来打开远程文件。

```
import urllib2
#调用 urllib2 中的 urlopen()
response=urllib2.urlopen('http://www.itzcn.com/')
方法打开远程文件
html=response.read()  #读取文件
print html
```

在该段代码中，首先使用 urllib2 模块中的 urlopen()方法打开了一个远程文件，返回一个类文件对象，该类文件对象支持 close()、read()、readline()和 readlines()方法，同时也支持迭代。在这段代码中，使用了该类文件的 read()方法来读取打开的远程文件源码。运行该段代码，在 Python 的解释器中输出 http://www.itzcn.com/路径所指向的页面源码，如图 11-6 所示。

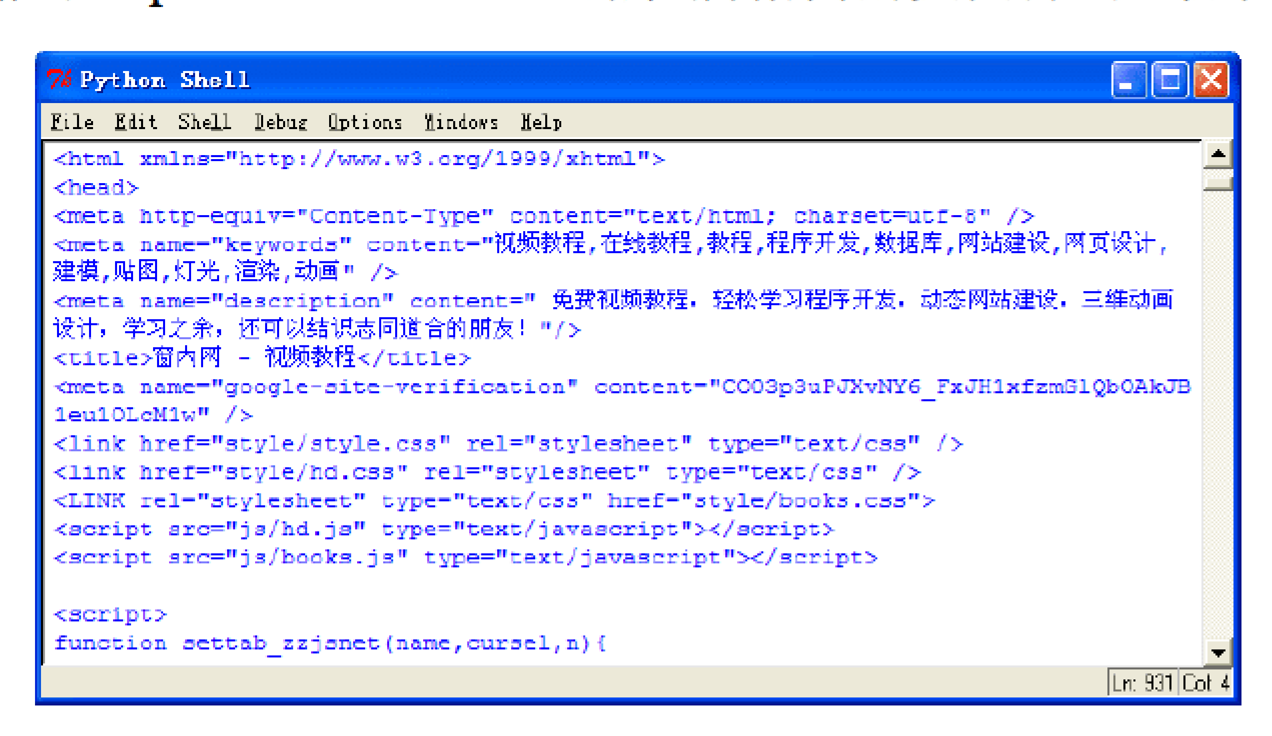


图 11-6　打开远程文件



如果需要访问本地文件，则可以用以 file 开头的 URL 作为 urlopen()方法的参数，比如 file://c:\\text\\somfile.txt，这里一定要对\进行转义。

2. 获取远程文件

方法 urlopen()提供一个能从中读取数据的类文件对象，如果需要 urllib 下载所指向的文件并在本地文件中存储一个文件的副本，那么就可以使用 urllib 模块中的 urlretrieve()方法。urlretrieve()方法返回一个元组(filename，headers)，而不是类文件对象。其中，filename 是本地文件的名字(由 urllib 自动创建)，headers 包含一些远程文件的信息。urlretrieve()方法的语法格式如下：

```
urlretrieve(url, filename=None, reporthook=None, data=None)
```

该方法的参数说明如表 11-2 所示。

表 11-2　urlretrieve()方法的参数说明

参数名称	描　述
url	符合 URL 规范的字符串
filename	本地文件路径的字符串，从 URL 返回的数据将保存在该文件中，如果设置为 None，则生成一个临时文件
reporthook	一个方法的引用，可以任意自定义该方法的行为，只需要保证方法有 3 个参数：第一个参数为目前为止传递的数据块数量；第二个参数为每个数据块的大小，单位为 byte；第三个参数为文件的总大小
data	向指定的 URL 发送的数据字符串，GET 和 POST 都可以，但必须符合标准格式，格式为：key=value

下面创建一个示例，具体介绍如何使用 urlretrieve()方法获取远程文件。

```
import urllib
# 使用 urllib 模块中的 urlretrieve()方法下载文件并保存至本地
urllib.urlretrieve('http://www.itzcn.com','D:\\python_webpage.html')
```

该段代码调用 urlretrieve()方法，下载 http://www.itzcn.com 的主页并存储至 D 盘下，文件名为 python_webpage.html。如果没有指定文件名，文件将会放到临时位置。用 open()方法可以打开该文件，完成对它的操作，可以删除它以节省硬盘空间。要清理临时文件，可以调用 urlcleanup()方法，但不要提供参数，该方法会负责清理工作。

11.2.3　基础知识——其他模块

前面讲过，除了本章提及的模块外，Python 库和其他地方还有很多与网络有关的模块，表 11-3 列出了 Python 标准库中的一些和网络相关的模块。

表 11-3　标准库中与网络相关的模块

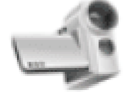
模　块	描　述
asynchat	asyncore 的增强版本
asyncore	异步套接字处理程序
cgi	基本的 CGI 支持
Cookie	Cookie 对象操作，主要用于服务器
cookielib	客户端 Cookie 支持
email	E-mail 消息支持(包括 MIME)
ftplib	FTP 客户端模块
gopherlib	Gopher 客户端模块
httplib	HTTP 客户端模块

续表

模　块	描　述
imaplib	IMAP4 客户端模块
mailbox	读取几种邮箱的格式
mailcap	通过 mailcap 文件访问 MIME 配置
mhlib	访问 MH 邮箱
nntplib	NNTP 客户端模块
poplib	POP 客户端模块
robotparser	支持解析 Web 服务器的 robot 文件
SimpleXMLRPCServer	一个简单的 XML-RPC 服务器
smtpd	SMTP 服务器端模块
smtplib	SMTP 客户端模块
telnetlib	Telnet 客户端模块
urlparse	支持解析 URL
xmlrpclib	XML-RPC 的客户端支持

11.3　服务器与客户端通信

服务器端和客户端的通信是构建网络通信的基础，例如 HTTP 访问。它也是一种客户端和服务器端的通信，即浏览器从 HTTP 服务器上获取 HTML 数据并显示出来。在本节中，将具体介绍服务器端的构建和客户端的访问方式。

**视频教学：**光盘/videos/11/服务器端和客户端通信.avi　　长度：11 分钟

11.3.1　基础知识——服务器端的构建

在讲解 Socket 模块的工作方式时，已经介绍了服务器端的通信技术，可以用其中所介绍的方法来构建服务器端应用程序。需要注意的是，当服务器端或者客户端关闭了连接后，服务程序将会重新等待下一个连接的到来。

1．一个小型服务器

下面来创建一个小型的服务器。

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)        #生成 Socket 对象
host=socket.gethostname()                                 #获取服务器名称
port=1234                                                 #端口号
s.bind((host,port))                                       #绑定 socket 地址
s.listen(5)                                               #开始监听
while True:
```

```
    c,addr=s.accept()                                    #接受一个连接
    print '连接来自: ',addr
    c.send('恭喜你! 一个简单的服务器创建成功! ')            #发送数据
    c.close()                                            #关闭连接
```

在代码的第一行，导入了 Socket 模块，这是任何使用套接字接口都需要做的工作，其中提供了套接字的大部分功能的实现。接着调用模块中的 socket()方法来生成 Socket 对象，下载的地址簇支持 3 种类型，包括 AF_INET、AF_INET6 和 AF_UNIX，默认值为 AF_INET。套接字类型包括 SOCK_STREAM 和 SOCK_DGRAM，或者其他 SOCK_常量，默认值为 SOCK_STREAM。协议值一般为 0。可以看出，代码片段中的 Socket 调用也可以简写成下面的形式：

```
s=socket.socket()
```

在生成套接字对象后，可以通过调用 bind()方法来绑定一个套接字地址，这里的 AF_INET 套接字地址为地址端口对。首先调用 gethostname()方法获取当前主机地址，并赋值给 host 变量。在 Socket 中，还提供了许多类似的实用方法。同时将 port 赋值为 1234。在 bind()方法中，注意到该方法只有一个参数，这个参数是一个地址端口对。



实用的端口号一般是被限制的。在 Linux 或者 UNIX 系统中，需要具有系统管理员的权限才能使用 1024 以下的端口号。这些低于 1024 的端口号被用于标准服务，比如端口 80 用于 Web 服务。如果停止了一个服务，可能需要过一段时间才能使用同一个端口号，否则会出现“地址正在使用”的错误信息。这里使用的是一个高于 1024 数值的端口号，在 bind()方法中绑定。

接着使用 listen()方法来使服务器进入侦听过程。listen()方法有一个参数，用来设置连接队列的长度。之后使用 while()循环，其条件 True 使得服务器进入死循环。也就是说，服务器端将一直处于侦听状态。使用 accept()方法可以接受客户端的一个连接，此对象返回一个元组，该元组有两个值：第一个值为生成的连接对象，可以使用此对象来发送和接收数据；第二个值为建立 Socket 连接的对端地址。接着使用 send()方法来向客户端发送一个字符串数据。最后调用 close()方法关闭此连接，从而释放此 Socket 连接所占用的资源。

2. 使用SocketServer模块

正如前面例子中使用 Socket 生成一个服务进程一样，比较复杂。为了能够简化服务器端的编程，Python 标准库提供了 SocketServer 模块。在此模块中有 5 个服务类，其关系如图 11-7 所示。

一般情况下，BaseServer 类不会被实际应用，因此 SocketServer 模块中只包含了 4 个基本的类：针对 TCP 套接字流的 TCPServer，针对 UDP 数据报套接字的 UDPServer，以及 UNIXStreamServer 和 UNIXDatagramServer。其中最重要的类是 TCPServer，此类中有简单的 TCP 协议实现的服务器接口，为应用程序提供了可靠的流数据传输，除此之外，还有更多模块中的类派生于 TCPServer 类，包括 BaseHTTPServer、SimpleHTTPServer、CGIHTTPServer、SimpleXMLRPCServer 和 DocXMLRPCServer 等。通过对 SocketServer 模块中类的继承，可以实现更多功能的服务器端应用程序。接着是 UCPServer 类，此类中有数据报服务的接口。此类同样提供数据传输服务，但是并不保证数据的可靠性和有序性。另外两个使用较少的是

UNIXStreamServer 和 UNIXDatagramServer 类，这两个类都是使用 UNIX 域套接字地址，它们几乎不能使用在非 UNIX 平台上。实际上，IP 和 UNIX 服务器端仅仅只是在套接字地址上不同而已。

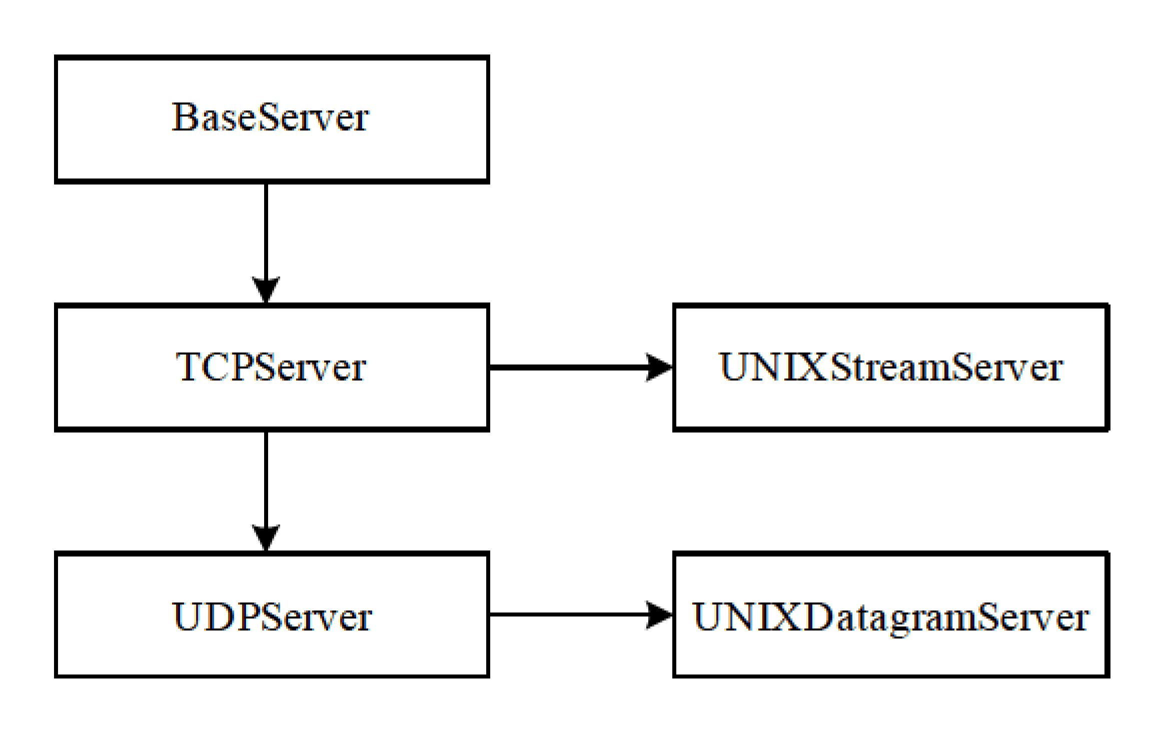


图 11-7　SocketServer框架图

为了写一个使用 SocketServer 框架的服务器，大部分代码会包含在一个请求处理程序(Request Handler)中。每当服务器收到一个请求(来自客户端的连接)时，就会实例化一个请求处理程序，并且它的各种处理方法会在处理请求时被调用，具体调用哪个方法取决于特定的服务器以及所使用的处理程序类，这样可以把它们子类化，使得服务器调用自定义的处理程序集。基本的 BaseRequestHander 类把所有的操作都放到了处理器的 handle()方法中，这个方法会被服务器调用，然后访问属性 self.request 中的客户端套接字。如果使用的是流(假若是 TCPServer)，那么可以使用 StreamRequestHander 类来创建两个新属性 self.rfile(用于读取)和 self.wfile(用于写入)，然后使用这些类文件对象和客户机进行通信。



SocketServer 框架中的其他类实现了对 HTTP 服务器的基本支持，其中包括运行 CGI 脚本，也包括对 XML RPC 的支持。

下面使用 SocketServer 模块制作一个小型服务器。

```
from SocketServer import TCPServer,StreamRequestHandler
class Handler(StreamRequestHandler):
    def handle (self):                              #重载处理方法
        addr=self.request.getpeername()             #获取连接对端地址
        print '获取的连接来自: ',addr
        self.wfile.write('恭喜你! 连接成功! ')      #发送数据
server=TCPServer(('',1234),Handler)                 #生成 TCP 服务器
server.serve_forever()                              #开始监听并处理连接
```

在代码的第一行，从 SocketServer 模块中导入 TCPServer 和 StreamRequestHandler，接着定义了 Handler 类，该类继承自 StreamRequestHander(StreamRequestHander 类专门用于 TCP 流服务)。在 Hander 类中，仅仅重载了 handle()方法，从而改变了其默认的处理方式。在此方法中有 3 条语句：

- 第一句通过 request 对象的 getpeername()方法获取连接对端的地址。
- 第二句将获取的连接地址打印出来。
- 第三句使用了 wfile()方法，向客户端写入一行信息。

接下来通过调用 TCPServer 类的构造函数生成一个服务程序对象，构造方法中有两个参数，第一个是套接字地址，第二个是连接的处理类。可以看到，这里的套接字地址与前面例子中是一样的。最后调用此服务程序对象的 serve_foreve()方法，开始监听并处理连接请求。调用这个方法使得其在被人工终止之前始终可以处理连接。实际上，这个方法只是简单地将此对象的 handle_request()方法放在一个无限循环中。



对于自定义的连接处理类，除了可以重载 handle()方法以外，还可以重载 setup()和 finish()方法。其中，前者是在 handle()方法之前做一些初始化工作，默认情况下不做任何处理。而 finish()方法则是在 handle()方法调用之后做一些连接的清理工作，默认情况下也是不做任何处理。当在 setup()或者 handle()方法中触发异常的时候，finish()方法将不会被调用。

另外，当连接到来时，handle()方法将会被调用。这时候，有些属性值可以被读取，例如 request 属性，这是一个连接请求对象，其值在流服务和数据报服务中是不一样的。在流服务中，其值为一个 Socket 对象，而在数据报服务中，其值仅仅只是一个字符串。除了 request 属性外，连接中还会提供其他的属性，例如通过 client_address 来表示客户端地址，通过 server 来表示一个服务器的实例。

对于每种具体的服务端接口，相应的类都提供了特定的方法来完成服务。通过重载某些方法，可以很快地实现不同的应用服务器程序。

11.3.2　基础知识——客户端的构建

相对于创建服务程序而言，创建一个客户端是比较简单的。在图 11-4 中，已经介绍了客户端创建的基本流程。下面来创建一个小型的客户机。

```
import socket
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)    #生成一个 Socket 对象
host=socket.gethostname()
port=1234
s.connect((host,port))                                #连接服务器
print s.recv(1024)                                    #读取数据
s.close()                                             #关闭连接
```

对一个客户端程序而言，首先需要将 Socket 模块导入，然后调用 Socket 模块中的 socket()方法生成一个 Socket 对象，并使用 gethostname()方法获取服务器地址，之后再通过 connect()方法连接服务程序。当客户端连接服务器后，一直等待的服务进程会被唤醒，并处理此连接。这里的客户端处理，直接调用 recv()方法获取服务器端发送过来的数据。最后调用 close()方法将连接关闭。运行该段代码，输出结果如下：

```
2011-03-30 14:33:22.128000
```

上面的例子创建了一个小型的客户机。其实，使用 Socket 模块和 urlparse 模块都可以创建一个简单的 HTTP 客户端，这个客户端并不具有浏览器的功能，只是获取指定 URL 的 HTML 文档内容，具体代码如下：

```
import socket
import urlparse
import sys
def httpServer (url):
    u=urlparse.urlparse(url)          #解析 URL
    host=u[1]                         #获取网络地址
    page=u[2]                         #获取页面地址
    s=socket.socket()                 #生成 socket 对象
    port=80                           #使用 80 端口号
    s.connect((host,port))            #连接服务器
    httpCmd='get'+page+'\n'
    s.send(httpCmd)                   #发送 HTTP 命令
    print s.recv(1024)                #获取内容
    s.close()                         #关闭连接
if  name  ==' main  ':
    #调用 httpServer()方法，传递 url
    httpServer('http://snaps.php.net/index.php')
```

该段代码通过使用 urlparse 模块中的 urlparse()方法，得到了此 URL 的网络地址和页面地址。接着使用 Socket 模块中的 socket()方法生成 Socket 对象，再使用该对象的 connect()方法开始连接服务器，在套接字地址的端口号中，使用了 HTTP 常用的 80 端口号。之后使用了 HTTP 的 get 命令来获取特定的 HTTP 页面，并使用 recv()方法获取返回的 HTML 文档内容。在 recv()方法中的参数 1024，使得这里只能最多读取 1024 字节的内容，该参数可以根据需要来调节大小，以获取所有的文档内容，并做进一步的处理。最后调用 close()方法关闭此连接。运行该段代码，输出结果如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
        <title>PHP Sources Snapshots</title>
        <style type="text/css" media="all">
        @import url("http://static.php.net/www.php.net/styles/site.css");
        @import url("http://static.php.net/www.php.net/styles/phpnet.css");
        </style>
        <!--[if IE]><![if gte IE 6]><![endif]-->
        <style type="text/css" media="print">
        @import url("http://static.php.net/www.php.net/styles/print.css");
        </style>
        <!--[if IE]><![endif]><![endif]-->
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <link rel="shortcut icon"
href="http://static.php.net/www.php.net/favicon.ico" />
    </head>
    <body>
        #省略以下代码
```



在上面的例子中，输出了指定的 URL 内容，实际上还有很多情况都没有考虑到，比如错误处理等。但是，毕竟已经实现了一种简单的获取内容的方法，其他具有此功能的库也是构建在这个基础之上的。

11.3.3　实例描述

使用 Socket 套接字不仅可以连接服务器，而且可以处理客户端的请求，从而达到服务器端与客户端的通信。下面我们就使用 Socket 对象来分别创建一个服务器端和客户端吧。

11.3.4　实例应用

【例 11-1】完成服务器端与客户端的通信工作。

(1) 新建 Python 文件，命名为 server.py，该文件用于创建服务器端。

(2) 生成 Socket 对象，并使用 bind()方法绑定主机 IP 和服务器进程所需的端口号，接着使用 listen()方法开始监听。之后进入 while()循环，使服务器端一直处于侦听状态。使用 accept()方法可以接受客户端的一个连接，接着使用 send()方法来向客户端发送一个字符串数据。最后调用 close()方法关闭此连接，从而释放此 Socket 连接所占用的资源。server.py 文件的内容如下：

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)    #生成 Socket 对象
sock.bind(('localhost', 8001))                  #绑定主机 IP 和端口号
sock.listen(5)                                  #开始监听
while True:
    connection,address = sock.accept()          #接收客户端的连接
    try:
        connection.settimeout(5)
        buf = connection.recv(1024)
        if buf == '1':
            connection.send('welcome to server!')     #向客户端发送一个字符串信息
        else:
            connection.send('please go out!')
    except socket.timeout:                      #如果出现连接超时的异常
        print 'time out'
    connection.close()
```

(3) 新建 Python 文件，命名为 client.py，该文件用于创建客户端。

(4) 使用 Socket 模块中的 socket()方法来生成 Socket 对象，然后使用 connect()连接服务器端，给予连接时间为 2 秒，并输出服务器端发送过来的信息，最后关闭 Socket 对象。client.py 文件的内容如下：

```
import socket
import time
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   #生成 Socket 对象
sock.connect(('localhost', 8001))         #连接服务器
time.sleep(2)
sock.send('1')                            #发送请求
print sock.recv(1024)                     #获取服务器端发送过来的信息
sock.close()
```

11.3.5 运行结果

在终端首先运行 server.py 文件，然后运行 client.py 文件，结果会在终端打印如下内容：

```
welcome to server!
```

11.3.6 实例分析



源码解析

在该案例中，分别创建了 server.py 文件和 client.py 文件。运行过 server.py 文件后，服务器将创建成功，服务器端的 IP 为 localhost，端口号为 8001。之后在 client.py 文件中使用了 connect()方法连接该服务器，从而输出服务器发送过来的信息 welcome to server!。

11.4 异步通信方式

在上面的示例中，所有的服务器端实现都是同步的。也就是说，服务程序只有处理完一个连接后，才能处理另外一个连接。如果要使服务器端应用程序能够同时处理多个连接，则需要使用异步通信方式。在 Python 标准库中有 3 种处理方式：分叉(ForkingMixIn)方式、线程(ThreadingMixIn)方式和异步 I/O(asynchronous I/O)方式。本节将详细介绍这 3 种处理方式。



视频教学：光盘/videos/11/异步通信方式.avi　　长度：7 分钟

11.4.1 基础知识——使用SocketServer进行分叉处理

当有多个连接同时到达服务器端的时候，可以通过分叉方式进行处理。对于接收到的每个连接，主进程都会生成相应的一个子进程专门用来处理此连接，而主进程则依旧保持在侦听状态，从而使每个连接都由一个对应的子进程来处理。由于生成的子进程和主进程是同时运行的，所以不会阻塞新的连接。这种方式的好处是处理比较简单、有效，但是由于生成进程消耗的资源比较大，这种处理方式当存在许多连接时，可能带来性能问题。



分叉是一个 UNIX 术语，当分叉一个运行的进程时，基本上就是复制了它。分叉后的两个进程都从当前的执行点继续执行，并且每个进程都有自己的内存副本。一个进程成为主进程，另一个(复制的)成为子进程。分叉操作在时间线上创建了一个分支，最后得到两个独立存在的进程。进程可以判断哪个是主进程哪个是子进程(通过查看 fork()方法的返回值)，因此它们所执行的操作是不同的。

在一个使用分叉的服务器中，每一个客户端连接都利用分叉创造一个子进程，主进程继续监听新的连接，同时子进程处理客户端。当客户端的请求结束时，子进程就退出了，因为分叉

的进程是并行运行的，客户端之间不必互相等待。下面编写代码来实现前面章节中案例的分叉处理：

```
from SocketServer import TCPServer, ForkingMixIn, StreamRequestHandler
class Server(ForkingMixIn, TCPServer):      #自定义 Server 类
    pass
class Handler(StreamRequestHandler):
    def handle (self):                       #重载 handle()方法
        addr=self.request.getpeername()
        print '获取的连接来自：',addr          #打印客户端地址
        self.wfile.write('使用 Fork 方式实现多连接')        #发送信息
if __name__=='__main__':
    server=Server(('localhost',1234),Handler)
    server.serve_forever()                   #开始侦听并处理连接
```

在该示例中，定义了一个 Server 类，该类继承自 ForkingMixIn 和 TCPServer 类，使此类具有这两个类的特点。也就是说，既提供流数据传输服务，又提供多连接处理的功能。其中 ForkingMixIn 类是需要第一个继承的类。同样，也可以结合 ForkingMixIn 和 UDPServer 类生成一个可以同时处理多个连接的数据报服务器端应用程序。在 Handler 类中，重载了 handle()方法，用于获取客户端地址并向客户端发送信息。

11.4.2　基础知识——使用线程方式

由于线程是一种轻量级的进程，具有进程所没有的优势。在上面讨论的分叉处理方式中，当存在大量连接而消耗资源太多的情况下，则可以使用线程方式进行处理。线程实现的方式和分叉的处理方式类似。当有连接到来的时候，主线程将生成一个子线程来处理连接，而在子线程处理连接的时候，主线程仍然处于侦听状态，并不会阻塞连接。

由于生成的子线程和主线程都存在于相同的进程中，共享内存，因此这种处理方式的效率非常高，从而使得大量地使用线程会造成线程之间的数据同步，如果处理不好，则可能使得服务程序时区效应，这就必须确保主线程和子线程的变量不冲突。在现代操作系统中(除了 Windows，它不支持分叉)，一般都使用分叉方式来处理多连接。



避免线程和分叉的另外一种方法是转换到 Stackless Python(http://stackless.com)，它是一个为了能够在不同的上下文之间快速、方便切换而设计的 Python 版本，它支持微线程的类线程并行形式，微线程比真线程的伸缩性要好，比如 Stackless Python 微线程已经被用在星战前夜在线(EVE Online，http://www.eve-online.com)来为成千上万的使用者服务。

下面创建一个示例，将上面的分叉处理示例修改为线程处理方式。

```
from SocketServer import TCPServer, ThreadingMixIn, StreamRequestHandler
class Server(ThreadingMixIn, TCPServer):
    pass
class Handler(StreamRequestHandler):
    def handle (self):                       #重载 handle()方法
        addr=self.request.getpeername()
        print '获取的连接来自：',addr          #打印客户端地址
        self.wfile.write('使用 Thead 方式实现多连接')
```

```
if __name__=='__main__':
    server=Server(('localhost',1234),Handler)
    server.serve_forever()
```

该段代码与使用分叉处理的示例相同。唯一区别在于，在生成的 Server 类时采用了 ThreadingMixIn 类，这样生成的 Server 类在处理多连接时将采用线程的方式来处理。

11.4.3　基础知识——异步IO方式

当服务器与客户端通信时，来自客户端的数据持续的时间较长且数据突发的多连接情况下，如果使用分叉或线程处理，则占用的资源太多。一种改进的方式就是采用专门的异步 IO 通信方式，即在一定的时间段内查看已有的连接并处理。处理的过程包括读取数据和发送数据。

在 Python 标准库中，由 asyncore 和 asynchat 模块来实现异步 IO 处理。这种功能依赖于 select() 和 poll()方法，这两个方法定义在 select 模块中。



关于 select()和 poll()方法，poll()方法的伸缩性更好，但它只能在 UNIX 系统中使用，在 Windows 系统中不可用。

1．select()方法

select()方法用于对指定的文件描述符进行监视，并在文件描述符集改变的时候做出响应。select 模块中的 select()方法的语法格式如下：

```
select.select(List rlist, List wlist, List xlist[, long timeout])
```

select()方法有 4 个参数：

- rlist、wlist 和 xlist。这是 3 个必需参数，分别表示等待输入、输出和错误的文件描述符，这 3 个参数都为文件描述符列表。空列表也是允许的，但是如果 3 个参数都是空列表，则表示平台相关。在 Linux 系统平台下允许，但在 Windows 系统平台下不允许，因为 Windows 中实现 TCP/IP 协议的 WinSock 不能处理文件描述符。
- timeout。这是一个可选参数，该参数为一个浮点数，用来指定系统监视文件描述符集改变的超时时间，单位为秒。当此参数被忽略的时候，方法将会阻塞到至少有一个文件描述符准备好的情况下才可返回。当设置超时的时间为 0 时，则表示调用时从来不会阻塞。

select()方法的返回值是一个包含 3 个值的元组，元组中的 3 个值即为在 select()方法中前 3 个参数已经准备好的文件描述符。当等待时间超过且没有任何已经准备好的文件描述符时，则返回由 3 个空列表组成的元组。

在文件描述符集合中，可以被接受的对象类型包括 Python 中的文件描述符，例如 sys.stdin 或者通过 open()和 popen()方法得到的对象，还包括通过 socket.socket()方法返回的 Socket 对象。这些可接受对象的一个共同特征是，都可以通过 fileno()方法来获取具体的描述符，一般使用整数来表示。下面创建一个示例，具体介绍 select()方法的使用。

```
import socket, select
s=socket.socket()                              #生成 Socket 对象
host=socket.gethostname()
port=1234
```

```
s.bind((host,port))                            #绑定套接字接口地址
s.listen(5)
inputs=[s]
while True:
    rs,ws,es=select.select(inputs,[],[])       #使用 select()方法
    for r in rs :
        if r is s:
            c,addr=s.accept()                  #处理连接
            print '获取连接来自: ',addr          #获取客户端地址
            inputs.append(c)
        else:
            try:
                data=r.recv(1024)              #接收数据
                disconnected=not data
            except socket.error:
                disconnected=True
            if disconnected:
                print r.getpeername(),'disconnected'
                inputs.remove(r)
            else:
                print data                     #打印接收到的数据
```

在该段代码中，首先导入了 Socket 模块和 select 模块，接着使用 Socket 模块中的 socket()方法来生成 Socket 对象，并使用 bind()方法绑定套接字接口地址和开始服务器端的监听。之后设置了一个 inputs 列表变量，用来记录需要处理输入的 Socket 对象。在 while 循环中，首先调用了 select()方法，该方法中的 3 个参数分别为 inputs 列表和两个空列表。由于没有超过时间的参数值，所以这里的调用将会一直阻塞到前 3 个集合中至少有一个文件描述符准备好为止，才能将返回值分别赋值给这 3 个变量，分别表示准备好的输入、输出和错误的文件描述符。可以看出，这里的 select()方法仅仅用于套接字的连接。实际上，如果加入对 sys.stdin 的监听，还可以实现对命令行输入数据的处理。



该例的功能仅仅是将收听到的数据打印出来，所以这里只考察了用于输入的文件描述符，而对已经准备好输入的每个文件描述符，这里仅考察所监听的输入情况。当确定好之后，代码调用 accept()方法来获取 Socket 对象和连接对端地址，然后决定将所生成的哪个 Socket 对象放到 inputs 列表中。

对于非监听端口的已准备输入的文件描述符，则通过调用 recv()方法来接收数据，当数据传输结束也就是没有数据时或者发生 Socket 错误时，设置连接断开标识。当连接断开时，打印相关的信息并将其连接对象从 inputs 列表中删除。

2．poll()方法

除了 Windows 平台外，poll()方法在其他系统下的应用十分广泛。该方法用于存在很多连接服务器的情况下。使用 poll()方法比 select()方法更简单，并且更容易扩展，原因在于 select()方法采用一种位图索引方式来处理文件描述符，而 poll()方法仅仅需要处理感兴趣的文件描述符即可，从而有效地降低了服务器的处理负担。

poll()方法同样是在 select 模块中实现的，在调用 poll()方法时，将得到一个 Polling 类对象。在生成的 Polling 类对象中，有 register()、unregister()和 poll()方法等。可以使用 Polling 类对象的 register()方法来注册一个文件描述符或者使用 fileno()方法得到的值。注册后可以使用 unregister()方法移除所注册的对象。当注册了一些对象(如 Socket 对象)以后，就可以调用 poll()

方法来获取一个(fd,event)对的列表，此方法有一个可选的超时参数。其中，fd 为文件描述符，event 用来指定发生的事件。event 是一个位掩码，通过一个整数位来对应特定的事件信息，表 11-4 是定义在 select 模块中的事件信息。为了验证一个指定的事件是否已经发生，可以使用&操作符，如下所示：

```
if event & select.POLLIN:
    #处理 POLLIN 事件
```

表 11-4　select模块中的poll()方法的事件常量

事件名称	描　述
POLLIN	含有可读的数据
POLLPRI	含有紧急可读的数据
POLLOUT	含有需要写出的数据
POLLERR	发生了错误
POLLHUP	连续断开
POLLVAL	错误的请求

下面使用 poll()方法重写前面使用 select()方法处理多连接的例子，即使用 poll()方法来代替 select()方法，代码如下：

```
import socket, select
s=socket.socket()                  #生成 Socket 对象
host=socket.gethostname()
port=1234
s.bind((host,port))                #绑定套接字接口地址
fdmap={s.fileno():s}
s.listen(5)                        #开始服务器端监听
p=select.poll()                    #生成 Polling 对象
p.register(s)                      #注册 Socket 对象
while True:
    events=p.poll()                #获取准备好的文件对象
    for fd,event in events :
        if fd is fdmap:
            c,addr=s.accept()      #处理连接
            print '获取连接来自：',addr
            p.register(c)
            fdmap[c.fileno()]=c    #加入连接 Socket
        elif event & select.POLLIN:
            data=fdmap[fd].recv(1024)
            if not data:           #没有数据
                print fdmap[fd].getpeername(),'disconnected'
                p.unregister(fd)   #移除注册
                del fdmap[fd]
        else:
            print data             #打印数据
```

在该段代码中，定义了一个字典变量 fdmap，用来保存需要监听的对象。接着使用 select 模块中的 poll()方法生成了一个 Polling 对象，并调用 register()方法注册了已经生成的 Socket 对象。在 while 循环中，包含对多连接的处理，首先调用 Polling 对象的 poll()方法，这里并没有

提供超时的参数，也就是说当代码运行到这里的时候，将阻塞直到有一个事件发生为止。判断返回的 fd 是否在 fdmap 字典中，如果存在，则表示此为监听连接的 Socket 对象，调用 accept()方法来获得客户端连接和客户端地址，然后将客户端地址输出，并注册客户端连接以及将其加入 fdmap 字典变量中。接着通过 event 和 POLLIN 标识位来判断是否为需要接收数据的事件，如果是需要接收数据的事件，则开始接收数据。当接收断开的连接时，则调用 unregister()方法取消此连接的注册信息，并将其从字典中移除。

11.4.4　基础知识——使用asyncore模块

在 Python 中，可以使用 asyncore 模块来实现异步通信。实际上，该模块提供了用来构建异步通信方式的客户端和服务器端的基础架构，特别适用于聊天类的服务器和协议的实现。其基本思想是，创建一个或者多个网络信道，实际上网络信道是 Socket 对象的一个封装，当信道创建后，通过调用 loop()方法来激活网络信道服务，直到最后一个网络信道关闭。

在 asyncore 模块中，主要用于网络事件循环检测的 loop()方法是其核心，在 loop()方法中将会通过 select()方法来检测特定的网络信道。当 select()方法返回有事件的 Socket 对象后，loop()方法检查此事件和套接字状态并创建一个高层次的事件信息，然后针对此高层次的事件信息调用相应的方法。asyncore 提供了底层的 API，用来创建服务器。

同时，在该模块中还有一个 dispatcher 类，这是一个对 Socket 对象的轻量级的封装，用于处理网络交互事件。其中的方法是在异步 loop()方法中调用，或者直接当作一个普通的非阻塞 Socket 对象，其框架如图 11-8 所示。

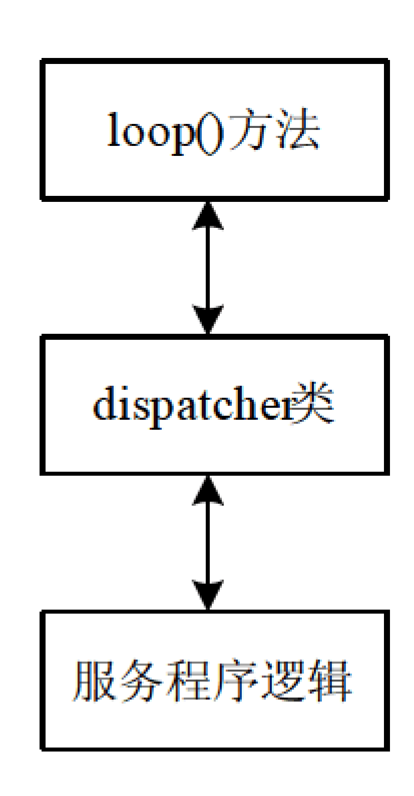


图 11-8　asyncore模块框架

在 dispatcher 类中，当在特定的时间或连续状态条件下，会触发一些高层次的事件。在继承类中可以通过重载这些方法来处理特定的事件，其默认事件如表 11-5 所示。

表 11-5　dispatcher类中定义的事件方法

方法名称	描　述
handle_connect	连接时的访问接口
handle_close	接口关闭
handle_accept	从监听端口上获取数据



在进行异步处理的过程中，可以通过网络信道的 readable()和 writable()方法来对事件进行控制。使用这两个方法，能够判断是否需要使用 select()或者 poll()方法来读取事件。

在 asyncore 模块中，readable()和 writable()方法默认不做任何操作，而直接返回 True，可以通过重载这两个方法来判断需要检查的连接，从而控制流程以及网络状态。之后，可以使用 handle_read()和 handle_write()方法来读写网络数据，完成对数据的接收和发送。在帮助手册中提供了一个基于 HTTP 协议的客户端例子，如下：

```
import asyncore, socket
class http_client(asyncore.dispatcher):    #定义了一个 http_client 类
    def __init__(self, host, path):        #类的构造函数
        asyncore.dispatcher.__init__(self)
        self.create_socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connect((host, 80))
        self.buffer = 'GET %s HTTP/1.0\r\n\r\n' % path
    def handle_connect(self):              #连接调用接口
        pass
    def handle_close(self):                #接口关闭方法
        self.close()
    def handle_read(self):                 #读取数据
        print self.recv(8192)
    def writable(self):                    #判断是否写入数据
        return (len(self.buffer) > 0)
    def handle_write(self):                #写入数据
        sent = self.send(self.buffer)
        self.buffer = self.buffer[sent:]
if __name__=='__main__':
    c = http_client(' snaps.php.net ', '/')
    asyncore.loop()
    print 'Program exit'                   #当执行完毕之后，输出字符串
```

在该段代码的开始部分，导入了 asyncore 模块和 Socket 模块，接着定义了一个名称为 http_client 的类，该类继承自 asyncore.dispatcher 类，因此可以在 http_client 类中重载 dispatcher 类中的处理方法。

在构造函数中，首先调用了 dispatcher 类的构造函数，即父类中的构造函数，接着调用 create_socket()方法来创建 Socket 对象，该方法封装了 Socket 模块中的 socket()方法，在调用 socket()方法之后，还使用了 setblocking()方法设置其阻塞方式为非阻塞，并获取了套接字的文件描述符。最后通过调用 add_channel()方法，将文件描述符加入。在构造函数中，使用 connect()方法连接特定服务器的 80 端口，这也是 HTTP 协议的默认端口，并设置类变量 buffer 为一个 HTTP 获取命令，此处构造的报文将在后面适当的时候被发送，用来获取 HTML 内容。

接下来定义了 5 个事件处理方法，分别在不同的事件发生时被调用。

- handle_connect()方法：将在 HTTP 连接的时候被调用。
- handle_close()方法：直接对 Socket 对象调用 close()方法，关闭连接；在 HTTP 关闭的时候被调用。
- handle_read()方法：调用 recv()方法来获取 HTTP 数据，会在获取数据的时候被调用。另外，recv()方法中的参数为一次最大读取的字节数。需要注意的是，缓冲区的大小

最好选择为 2 的幂，如 1024 或者 4096 等数据。

- handle_write()方法：用来处理发送时的数据。这里首先调用 send()方法发送数据，其返回值为已经发送成功的数据，然后设置 buffer 为未发送的数据。这样做的原因是在异步通信过程中，不一定能够保证每次发送都能发送成功。
- writable()方法：用来判断在什么时候发送数据。在方法体中，只是判断需要发送数据的缓冲区是否为空，如果不为空，则返回 True，表示需要发送数据，而当缓冲区为空时，则不需要继续发送数据。

在调用 http_client 类时，首先会执行 handle_connect()方法，当连接成功后，会立刻执行 writable()方法，默认的 writable()方法直接返回 True。这里进行了重载，用于对缓冲区进行判断。当发现缓冲区不为 0 时，就会返回 True，直到缓冲区变成 0，才返回 False。当 writable()方法返回 True 时，就会触发 handle_write()方法。该方法使用 send()发送数据，并检查第一次发送信息的长度，以便对缓冲区进行截取，留下未发送完的数据。这样 writable()方法检查到缓冲区还有数据，又会返回 True，触发 handle_write()方法，直到把缓冲区的数据全部发送完毕为止。send()方法的具体实现如下：

```
def send(self, data):
  try:
     result = self.socket.send(data)
     return result
  except socket.error, why:
     if why[0] == EWOULDBLOCK:
        return 0
     else:
        raise
     return 0
```

在该段代码的最后，使用 http_client 类生成一个实例，最后调用 asyncore 模块中的 loop()方法。运行该段代码，输出结果如下：

```
HTTP/1.1 200 OK
Date: Sun, 03 Apr 2011 08:10:38 GMT
Server: Apache/1.3.37 (Unix) PHP/5.3.3-dev
X-Powered-By: PHP/5.3.3-dev
Connection: close
Content-Type: text/html; charset=utf-8

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
                  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
   <head>
      <title>PHP Sources Snapshots</title>
      <style type="text/css" media="all">
      @import url("http://static.php.net/www.php.net/styles/site.css");
      @import url("http://static.php.net/www.php.net/styles/phpnet.css");
      </style>
      <!--[if IE]><![if gte IE 6]><![endif]-->
      <style type="text/css" media="print">
      @import url("http://static.php.net/www.php.net/styles/print.css");
      </style>
      <!--[if IE]><![endif]><![endif]-->
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
      <link rel="shortcut icon"
href="http://static.php.net/www.php.net/favicon.ico" />
```

```
    </head>
    <body>
#省略部分 HTML 代码
</body>
</html>
Program exit
```

这里，不仅仅输出了 HTML 文档的全部内容，而且输出了 HTTP 响应的内容。输出结果中的 HTTP/1.1 200 OK 表名此次连接的获取是正常的，然后输出所获取的 HTML 文档的信息，例如最后一次修改的时间、文档的长度和文档的类型等。作出 HTTP 响应后，输出获取到的 HTML 文档内容。

11.4.5　实例描述

在实际应用中，一个应用程序很少会固定等待 Socket 连线，或者传送资料，通常都是在突发情况下一个 Socket 要进行连线，或者要传送资料。非同步的 Socket 的好处就在这里，你可以指定在 Socket 发生连线或者传送资料的时候进行某个动作，而 asyncore 就是把这样的非同步行为封装成一个类，让我们只要继承它就能达到这样的效果。

下面创建一个案例，该案例主要用途在于客户端连接服务器端，将服务器端发送的信息打印出来。

11.4.6　实例应用

【例 11-2】使用 asyncore 和 socket 模块将所请求的服务器端信息打印出来。

(1) 新建 Python 文件，命名为 http_client.py。

(2) 将 asyncore 和 Socket 模块导入，代码如下：

```
import asyncore,socket
```

(3) 新建 AsyncGet 类，该类继承自 asyncore.dispatcher 类，并重载 asyncore.dispatcher 类中的__init__()、handle_connect()、handle_read()、writeable()、handle_write()和 handle_close()方法。在__init__()方法中获取与服务器端的连接，并将指定的页面请求赋值与 request 对象；在 handle_connect()方法中输出连接的服务器信息；在 handle_write()方法中将请求对象 request 发送到服务器端；在 handle_read()方法中将获取的服务器发送给客户端的信息写入一个记事本中。AsyncGet 类的代码如下：

```
class AsyncGet(asyncore.dispatcher):
    def __init__(self,host):
        asyncore.dispatcher.__init__(self)
        self.host=host
        #创建 Socket 对象
        self.create_socket(socket.AF_INET, \
                           socket.SOCK_STREAM)
        self.connect((host,80))                    #连接服务器
        self.request='GET /index.html HTTP/1.0\r\n\r\n' #请求 index.html 页面
        self.outf=None
        print '请求的 index.html 来自：',host
```

```
    def handle_connect(self):
        print '连接：',self.host
    def handle_read(self):
        if not self.outf:
            print '正在创建连接：：',self.host
        self.outf=open('%s.txt'%self.host,'wb')  #将服务器信息写入记事本中
        data=self.recv(8192)                      #获取服务器发送过来的信息
        if data:
            self.outf.write(data)                 #写入记事本中
    def writeable(self):
        return len(self.request)>0
    def handle_write(self):
        num_sent=self.send(self.request)          #发送客户端请求
    def handle_close(self):
        asyncore.dispatcher.close(self)
        print 'Socket 对象关闭于：',self.host
        if self.outf:
            self.outf.close()
```

(4) 编辑主程序入口，实例化 AsyncGet 类，代码如下：

```
if __name__=='__main__':
    AsyncGet('www.python.org')
    asyncore.loop()
```

11.4.7　运行结果

运行 http_client.py 文件，打印连接服务器的信息，如图 11-9 所示。打开 http_client.py 文件所在的目录，将会发现在该目录下多了一个 www.python.org.txt 文件，该文件记录了服务器发送给客户端的所有信息，比如服务器名称、内容长度、内容类型等。

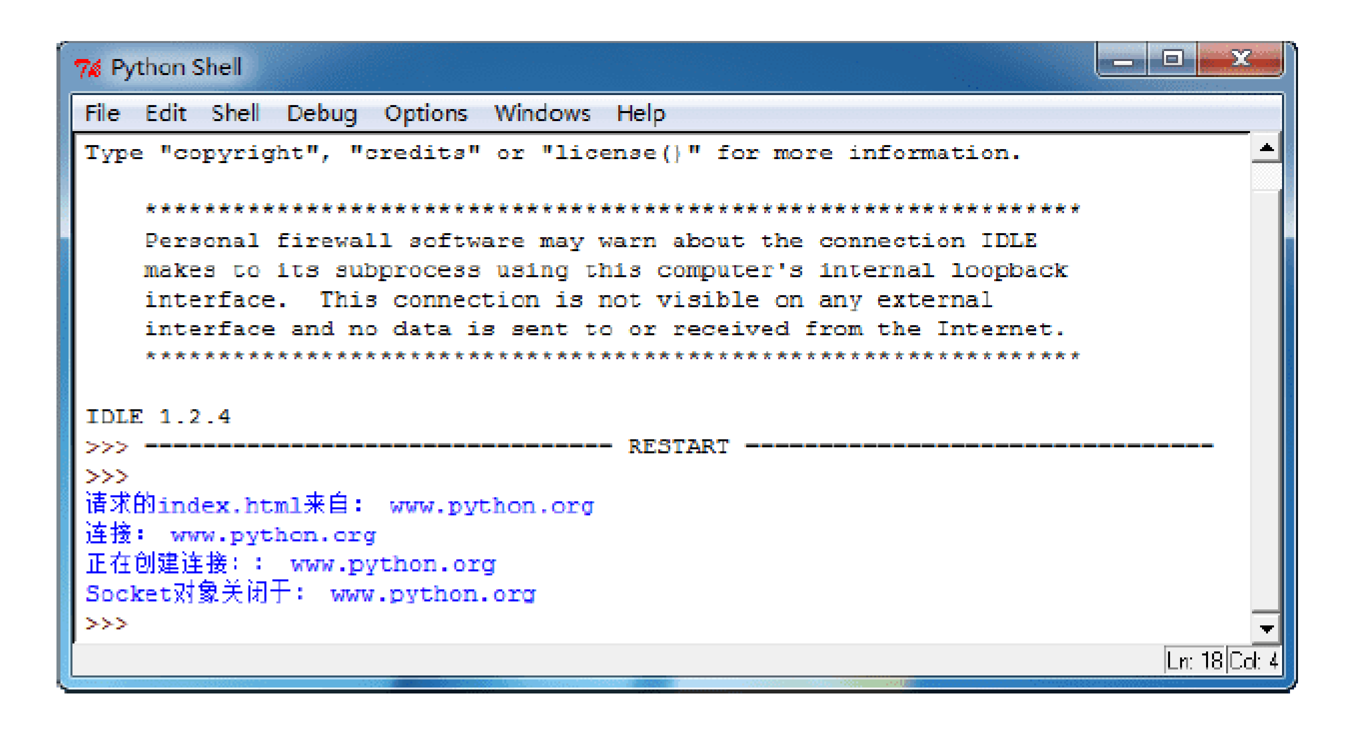


图 11-9　打印连接服务器的信息

11.4.8　实例分析



源码解析

在该案例中，创建了 asyncore.dispatcher 的子类，并重载了该类中的许多方法。在__init__()

方法中，使用 create_socket()方法创建了 Socket 对象，并使用 connect()方法连接服务器，端口号为 Web 开发最常用的 80 端口。在本案例中，所连接的服务器为一个远程的网络地址 www.python.org。当运行该程序时，系统将访问 www.python.org/index.html 页面，并打印连接服务器的信息。

11.5 实现一个简单Web服务器

Twisted 是一个面向对象基于事件驱动的顶级通信框架，它允许使用和开发完全异步的网络应用程序和协议。同时，Twisted 框架具有很好的网络性能，提供了异步通信机制。在本节中，将在介绍 Twisted 网络框架的基础上，对如何使用它来构建网络服务器端进行介绍。

 **视频教学：**光盘/videos/11/ Twisted 网络框架.avi　　长度：7 分钟

11.5.1 基础知识——初始Twisted框架

来自于 Twisted Matrix 实验室的 Twisted 是一个事件驱动的 Python 网络框架，已经应用于多个领域。Twisted 是 ZOPE 中 HTTP 服务器的实现部分，可以和大名鼎鼎的 ACE(Adaptive Communication Environment，自适应网络通信环境)网络框架媲美。它特别适合于用来编写服务器端的应用程序，对于其中的很多细节，Twisted 都有比较完美的实现。

在 Twisted 框架中，已经提供了许多可重用的协议和接口。这些协议包括 SSH2、FTP、POP3 和 SMTP 等，甚至还有对 MSN 等即时通信协议的支持。

11.5.2　基础知识——下载并安装Twisted

安装 Twisted 很简单，步骤如下。

(1) 首先需要访问 Twisted Matrix 的主页 http://twistedmatrix.com，如图 11-10 所示。

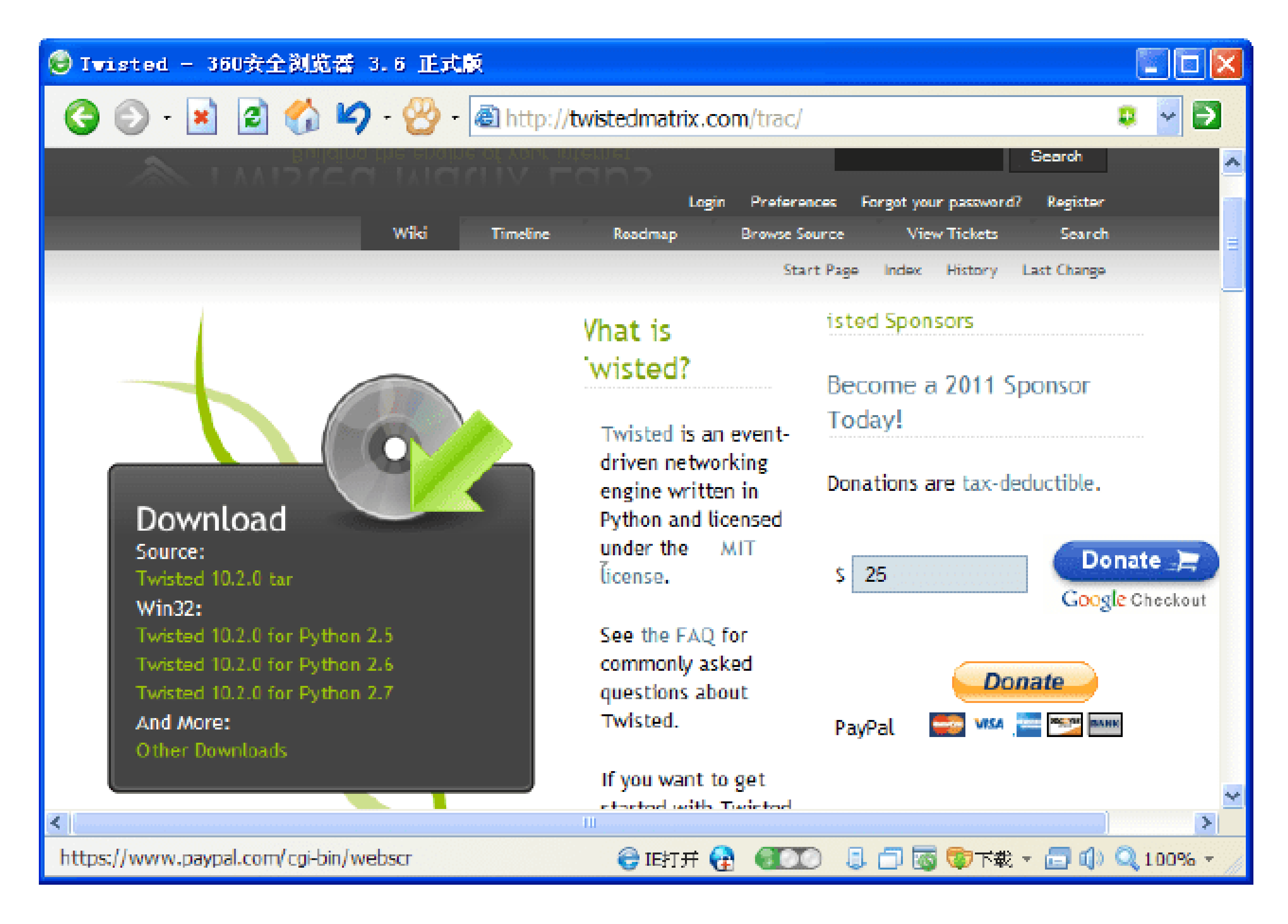


图 11-10　Twisted主页

(2) 单击 Download 栏目中的下载连接。如果使用的是 Windows 系统，则下载与 Python 版本对应的 Windows 安装程序；如果使用的是其他系统，则下载源代码档案文件(如果使用了包管理器，比如 Portage、PRM、APT、Fink 或者 MacPorts，那么可以直接下载并安装 Twisted)。这里单击 Win32 目录下的 Twisted 10.2.0 for Python 2.5 进行下载。

(3) 双击下载好的 Twisted-10.2.0.winxp32-py2.5.msi 安装文件，进入 Twisted 网络框架的安装程序，如图 11-11 所示。

(4) 这里选择 Install just for me 单选按钮，然后单击 Next 按钮，进入选择安装目录界面。这里选择 Python 的安装目录，如图 11-12 所示。

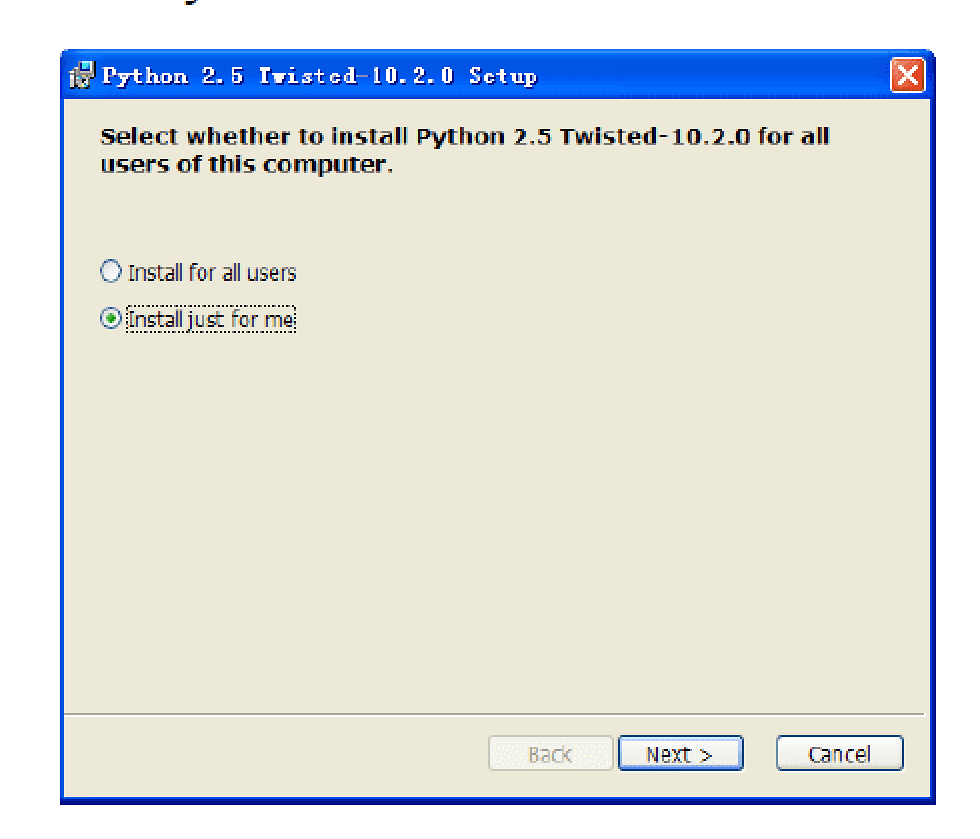


图 11-11　Twisted的安装主界面

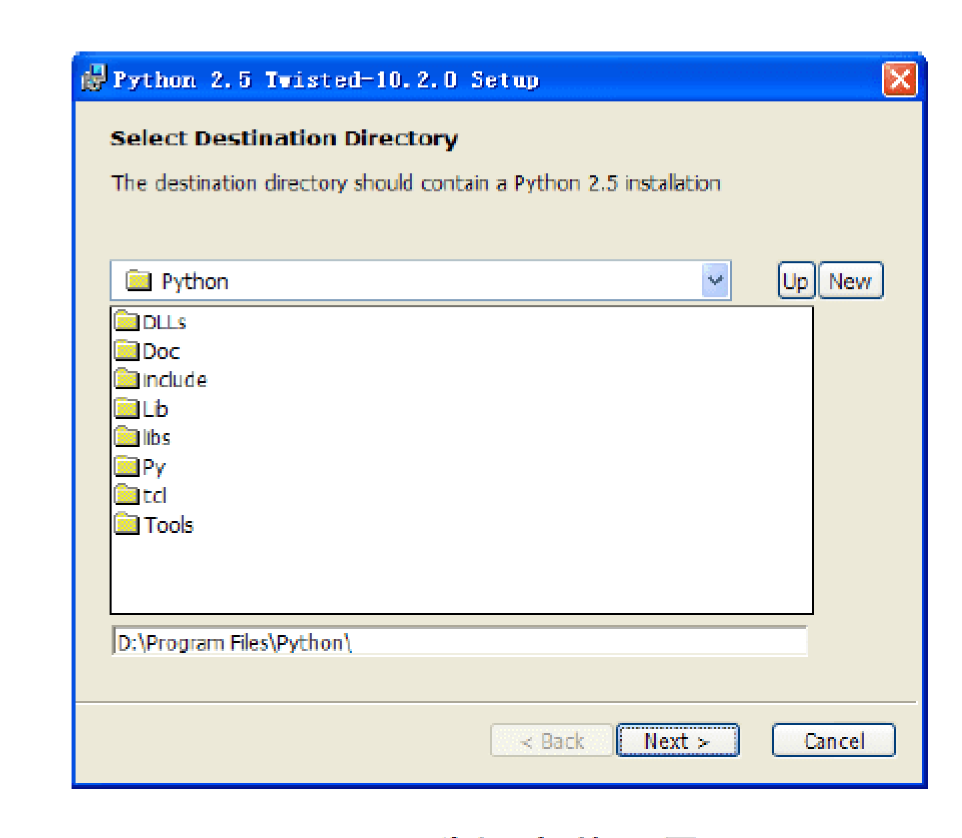


图 11-12　选择安装目录界面

(5) 单击 Next 按钮，进入 Twisted 安装界面，如图 11-13 所示。安装完成后，自动转到安

装结束界面，如图 11-14 所示。

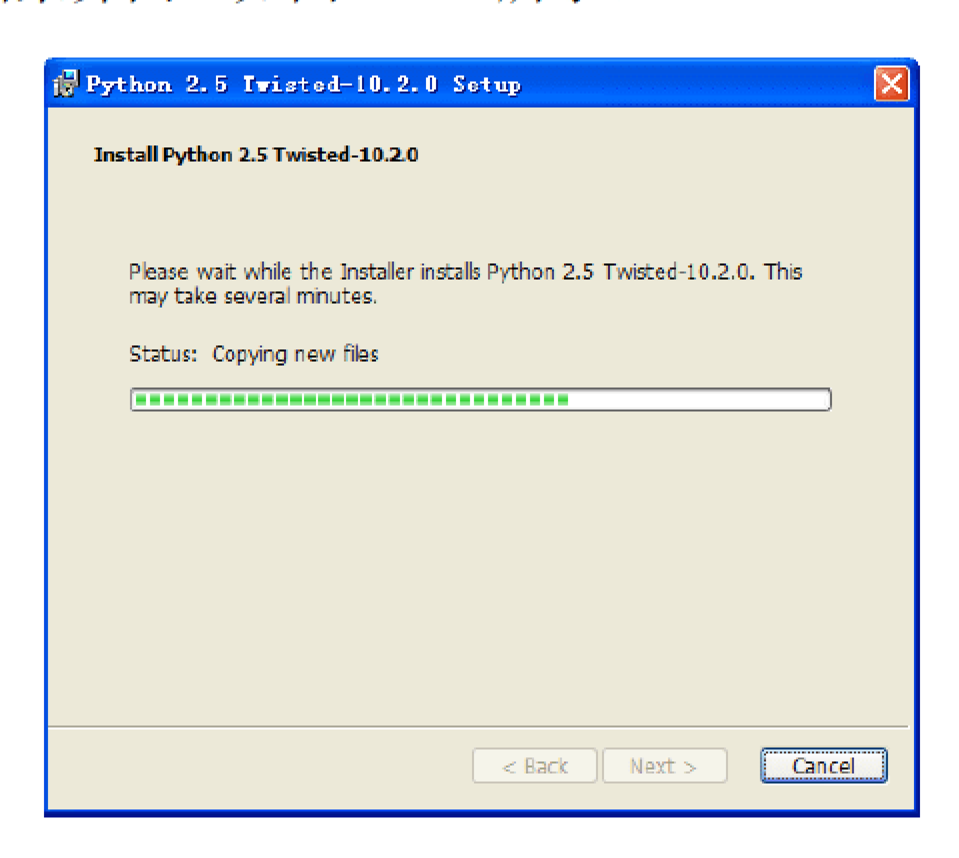


图 11-13　Twisted安装界面

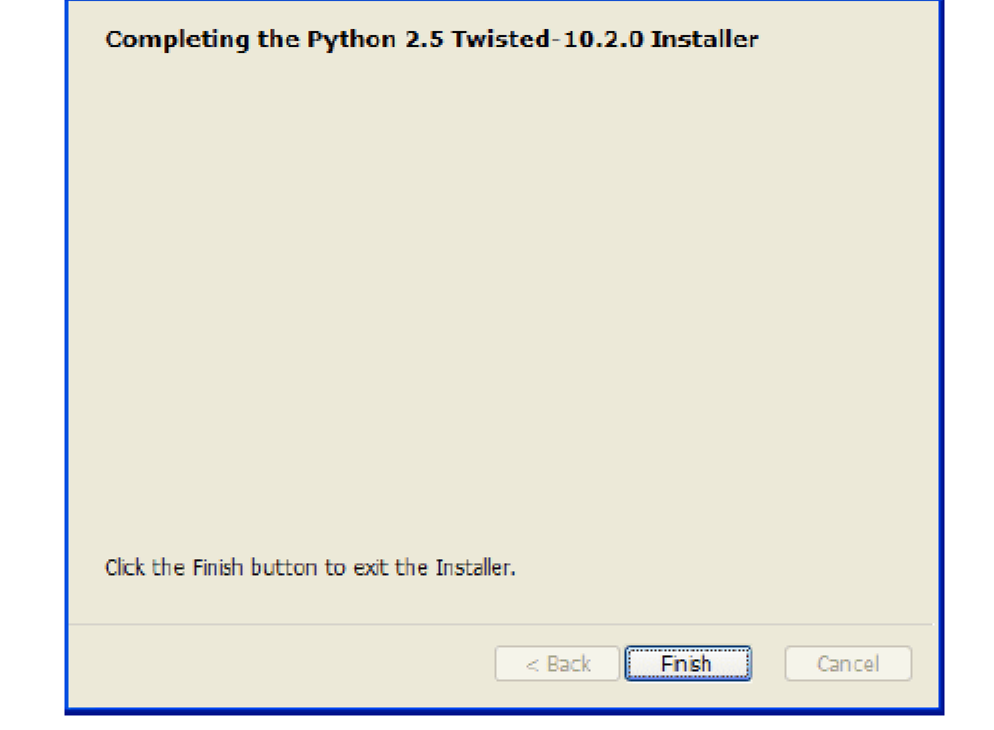


图 11-14　安装完成界面

(6) 单击 Finish 按钮，安装完毕。安装完成后，在 Python 安装目录下的 site-packages 目录下将会生成一个 twisted 目录。

11.5.3　基础知识——编写Twisted服务器

在使用分叉和线程方式来进行异步通信时，实质上仍然采用了一种轮询方式来处理连接。对 Twisted 框架(包括前面所讲到的 asyncore 模块)来说，采用的是一种事件驱动的方式。也就是说，只需要在事件发生的点构建相应的代码就可以了。例如构建一个服务器端应用程序，只需要在部分事件的接口上书写代码即可，包括新连接到来的时候、新数据到来的时候和连接关闭的时候等。Twisted 框架做了进一步封装。例如，不仅仅只针对新数据到来时进行处理，而是将这样的事件分解成更基本的事件，包括换行之前数据的到来事件等。

事件处理程序在一个协议中定义。在一个新的连接到来时，同样需要一个创建这种协议对象的工厂，如果只想创建一个通用的协议类的实例，那么就可以使用 Twited 自带的工厂。Factory 类在 twisted.internet.protocol 模块中。当编写自己的协议时，要使用和超类一样的模块中的 protocol。得到一个连接后，事件处理程序 connectionMade 就会被调用；而丢失一个连接后，connectionLost 就会被调用。来自客户端的数据是通过处理程序 dataReceived 接收的。当然不能使用事件处理策略来把数据发回客户端，如果要实现此功能，可以使用对象 self.transport，这个对象有一个 write()方法，也有一个包含客户机地址(主机名和端口号)的 client 属性。下面创建一个在 Twisted 框架下服务器端的实现示例。

```
from twisted.internet import reactor
from twisted.internet.protocol import Protocol,Factory
class SimpleLogger(Protocol):
    def connectionMade (self):                  #连接建立时
        print '获取的连接来自：',self.transport.client
    def connectionLost (self,reason):           #连接断开时
        print self.transport.client,'disconnected'
    def dataReceived (self,data):               #接收数据时
        print data
factory = Factory()
```

```
factory.protocol=SimpleLogger
reactor.listenTCP(1234,factory)
reactor.run()                                    #进入循环
```

在 Twisted 框架下创建一个服务器，必须实例化 Factory 类，同时需要设置它的 protocol 属性，这样当它和客户机通信时就知道使用什么协议，然后开始在指定的断口处使用工厂进行监听，这个工厂要通过实例化协议对象来准备处理连接。程序使用的是 reactor 中的 listenTCP()方法来监听，最后通过调用同一个模块中的 run()方法启动服务器。运行该段代码，将打印错误信息，如下所示：

```
Traceback (most recent call last):
  File "14.py", line 1, in <module>
    from twisted.internet import reactor
  File "D:\Program
Files\Python\lib\site-packages\twisted\internet\reactor.py", line 37, in
<module>
    from twisted.internet import selectreactor
  File "D:\Program
Files\Python\lib\site-packages\twisted\internet\selectreactor.py", line 17,
in <module>
    from zope.interface import implements
ImportError: No module named zope.interface
```

从以上错误信息可以看出，缺少 zope.interface 模块。在这种情况下，从网上下载 zope.interface-3.6.1-py2.7-win32.zip 压缩包，解压缩为 zope 文件夹，将该文件夹放入 Python/Lib/site-packages 目录下。再次运行该段代码，服务器创建完成。

11.5.4　实例描述

在 PHP 程序中，如果需要访问一个名称为 index.php 的页面，则需要在浏览器的地址栏中输入 http://localhost:80/index.php。其中，localhost 表示为本地 IP 地址，80 为 PHP 服务器的端口号，这里的 PHP 服务器并不是自定义的，需要将该服务器安装于本地电脑上，这将占用电脑的一部分空间。因此，我们可以自定义 Web 服务器。下面使用 Twisted 框架来自定义一个 Web 服务器。

11.5.5　实例应用

【例 11-3】使用 Twisted 实现一个简单 Web 服务器。

(1) 在项目的根目录下新建 htm 文件夹，并在该目录下新建 web.html 文件，内容根据需要自定。

(2) 在 htm 文件夹的同级目录下新建 web.py 文件。

(3) 将 Resource 类、server 类、static 类和 reactor 类导入 web.py 文件中，并定义全局变量 PORT，存储服务器连接的端口号，代码如下：

```
from twisted.web.resource import Resource
from twisted.web import server
from twisted.web import static
from twisted.internet import reactor
```

```
PORT=1234
```

(4) 继续在 web.py 文件中创建 ReStructured 类，该类继承自 Resource 类，用于读取指定的源文件内容，代码如下：

```
class ReStructured( Resource ):
  def __init__( self, filename, *a ):
      self.rst = open( filename ).read( )            #打开指定文件并读取
  def render( self, request ):
      return self.rst
```

(5) 指定将要读取的源文件，并调用 ReStructured 类进行处理，代码如下：

```
resource = static.File('htm/')                       #指定要读取文件所在的路径
resource.processors = { '.html'  : ReStructured }   #指定处理器
resource.indexNames = [ 'index.html']                #指定要读取文件的文件名
```

(6) 调用 reactor 类中的 listenTCP()方法，设置监听的端口号和工厂对象。最后使用 reactor 类的 run()方法进入循环。在此期间，程序将监听 1234 端口，当收到连接请求时，将会调用 ReStructured 类来处理特定的事件。代码如下：

```
reactor.listenTCP(
      PORT,
      server.Site( resource )
      )
reactor.run()
```

11.5.6 运行结果

先运行 web.py 文件，无任何输出，然后打开浏览器，输入 http://localhost:1234/，按回车键，将会访问 htm 文件夹下的 index.html 文件(在 htm 文件夹下并没有定义 index.html 文件，但是程序将会自动生成该文件，该文件将显示出 index.html 文件所在的站点目录)，如图 11-15 所示。

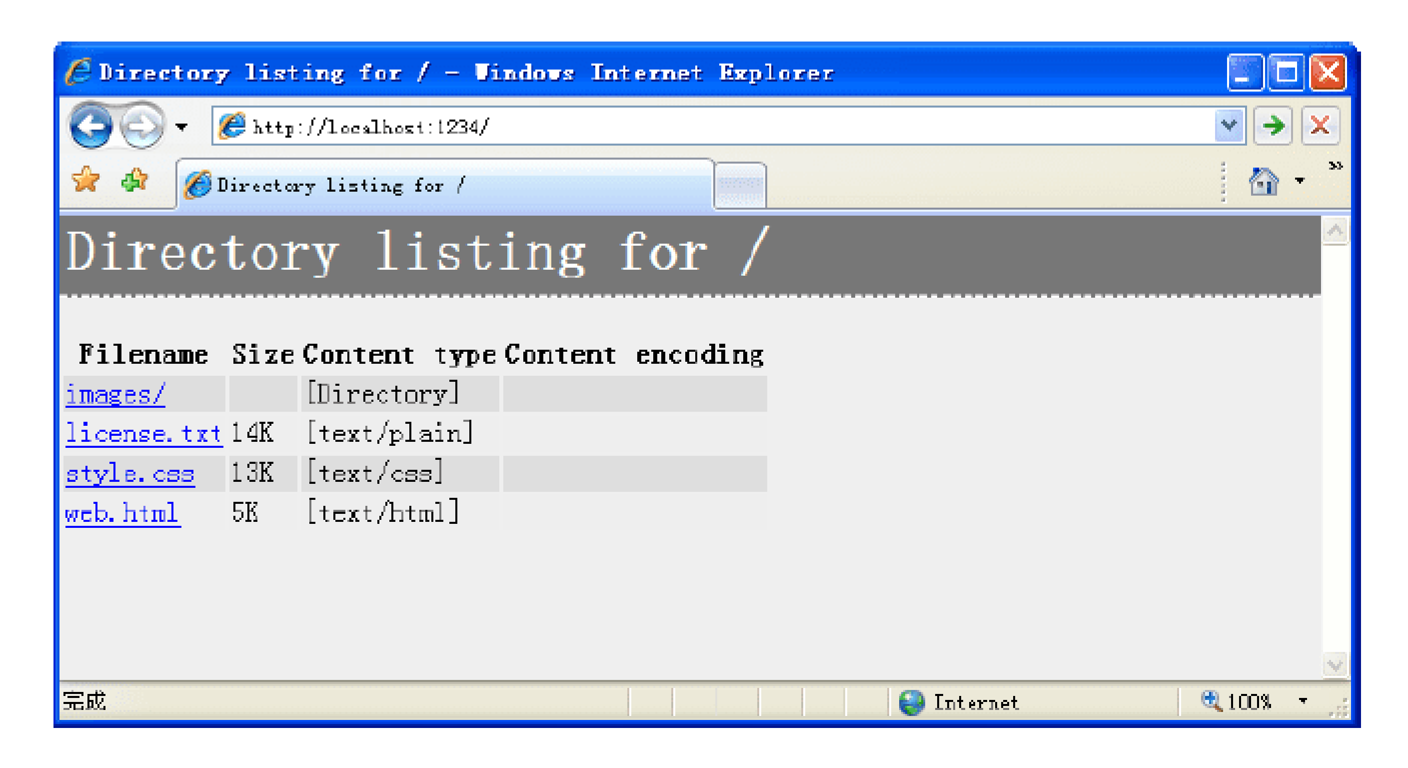


图 11-15 自定义服务器

11.5.7　实例分析



源码解析

该案例实现了一个使用 Twisted 创建 Web 服务器的功能。当访问 http://localhost:1234/时，监听 1234 端口并调用 ReStructured 类对请求作出处理。在 HTTP 中请求报文处理对象为 twisted.web.http.Request 类，而每个资源都是 twisted.web.resource.Resource 的子类，该案例中的 ReStructured 类继承自 Resource 类，并重载了 render()方法来响应请求，即在 htm 文件夹下创建 index.html 页面。该方法中的 request 对象实际就是 Request 类的实例。

11.6　常见问题解答

11.6.1　Python Socket编程疑问



Python Socket 编程疑问？

网络课堂：http://bbs.itzcn.com/thread-15819-1-1.html

运行一个官方的例子：

```
import socket
HOST = ''                 # Symbolic name meaning all available interfaces
PORT = 50007              # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
s.listen(1)
conn, addr = s.accept()
print 'Connected by', addr
while 1:
    data = conn.recv(1024)
    if not data: break
    conn.send(data)
conn.close()
```

每次运行到 data = conn.recv(1024)这句话时卡住，如果在其后加上 break 后就好了。我用浏览器往这个程序里发送过 GET 请求，可没什么反应，然后将这个程序强制关闭，浏览器就接收到数据了，这说明第一次接收数据是成功的，第二次应该返回 0，然后执行 break，又被卡住了，这是怎么回事呢？

【解决办法】其实，这个问题应该是众多初学者都会遇到的，例子本身没有问题。在 while 循环中，只要浏览器和服务器(你编写的程序)仍然保持通信，循环就不结束，因此会存在卡住的现象，其实是因为服务器端一直等待接受客户端发送数据，所以只要你强制关闭服务器端，这个 while 循环才会退出。通常，当发送很长的请求到服务器时，需要使用某种方法表示所发送的数据长度。

11.6.2 Pydev调用Twisted模块的reactor错误



Pydev 调用 Twisted 模块中的 reactor 时错误！
网络课堂：http://bbs.itzcn.com/thread-15820-1-1.html

在 Eclipse 中新建一个 Pydev Module，内容如下：

```
from twisted.internet import reactor
import time
def printTime():
    print "当前时间为：",time.strftime("%H:%M:%S")
def stopReactor():
    print "响应停止"
    reactor.stop()
reactor.callLater(1,printTime)
reactor.callLater(2,printTime)
reactor.callLater(3,printTime)
reactor.run()
```

运行也正常，但是在代码的第一行前总是有一个红叉，提示找不到 reactor 类，这是怎么回事？

【解决办法】这是因为 twisted 模块和 Pydev 中的 twisted 模块重名所致，只需要在 Eclipse 开发工具中，选择 Window | Preferences | Pydev | Interpreter-Python | Forced Buildins 选项卡，单击右边的 New 按钮，然后在输入框中输入 twisted，再单击 OK 按钮即可，这样就导入了 twisted 模块。

11.7 习　　题

一、填空题

(1) __________模块提供了对客户端和服务器端套接字的低级访问功能，服务器端套接字会在指定的地址监听客户端的连接，而客户机则是直接连接服务器的。

(2) 在 asyncore 模块中，主要用于网络事件循环检测的__________方法是其核心，在该方法中将会通过 select()方法来检测特定的网络信道。

(3) __________是一个同步的网络服务器基类，它位于 Python 标准库中，使用它可以很容易地编写服务器。它甚至用 CGI 支持简单的 Web 服务(HTTP)。

二、选择题

(1) __________和__________模块可以在给出数据源的 URL 时，允许用户从不同的服务器端读取和下载数据，两者都可以通过 urlopen()方法来工作。

A. socket urllib　　　　B. urllib urllib2
C. urllib1 urllib2　　　　D. twisted urllib

(2) 运行下面代码，输出的结果为：__________。

```
import urllib2
```

```
    #调用 urllib2 中的 urlopen()方法，打开远程文件
response=urllib2.urlopen('http://www.itzcn.com/')
html=response.write()  #读取文件
print html
```

A. 输出 http://www.itzcn.com 的主页源代码

B. 什么也不输出

C. 打印错误信息

D. 显示 http://www.itzcn.com 的主页

(3) ＿＿＿＿＿＿支持绝大多数的网络协议，它内容丰富并且很复杂。该框架是异步的，因此它在伸缩性和效率方面表现得很好，它可能是很多自定义网络应用程序的最佳选择。

A. SocketServer 框架　　　　B. Socket 框架

C. Twisted 框架　　　　　　D. asyncore 框架

三、上机练习

上机练习： 使用 Python 中的 urllib 模块从服务器下载指定文件到本地。

实用 urllib 模块中的 urlretrieve()方法直接将远程服务器的数据下载到本地。该方法有 3 个参数：第一个参数指定要下载数据的网络路径；第二个参数为保存到本地的路径(如果未指定该参数，urllib 会生成一个临时文件来保存数据)；第三个参数为代理服务器地址。该方法返回一个包含两个元素的元组(filename，headers)。其中，filename 表示保存到本地的路径，header 表示服务器的相应头。检测要下载文件的大小，如果没有下载完成，则打印“正在下载，请耐心等待……”，如图 11-16 所示；否则打印“下载完成……”，如图 11-17 所示。下载完成后，在 urlretrieve()方法中的第二个参数所指定的目录下存在下载好的远程数据。

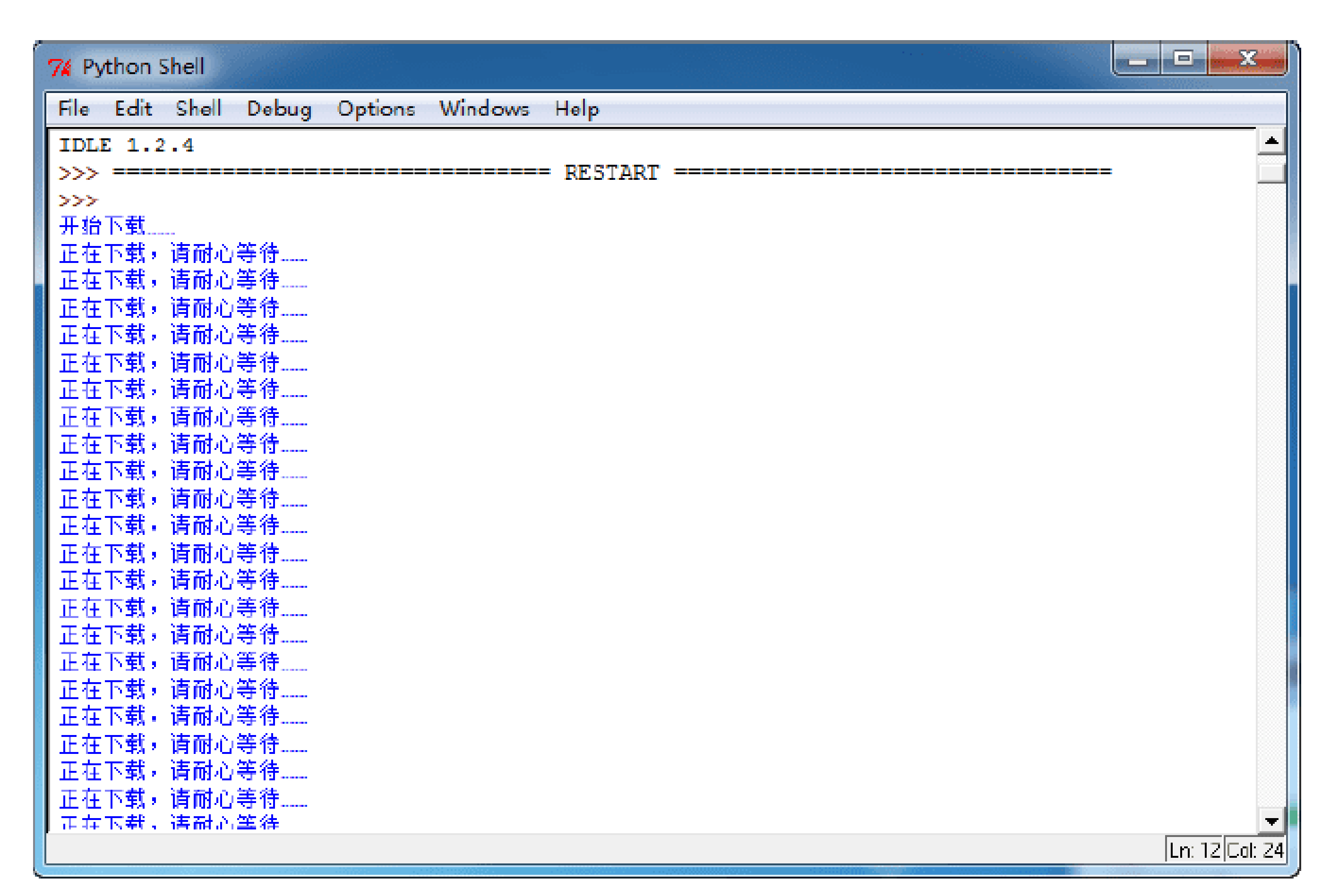


图 11-16　正在下载

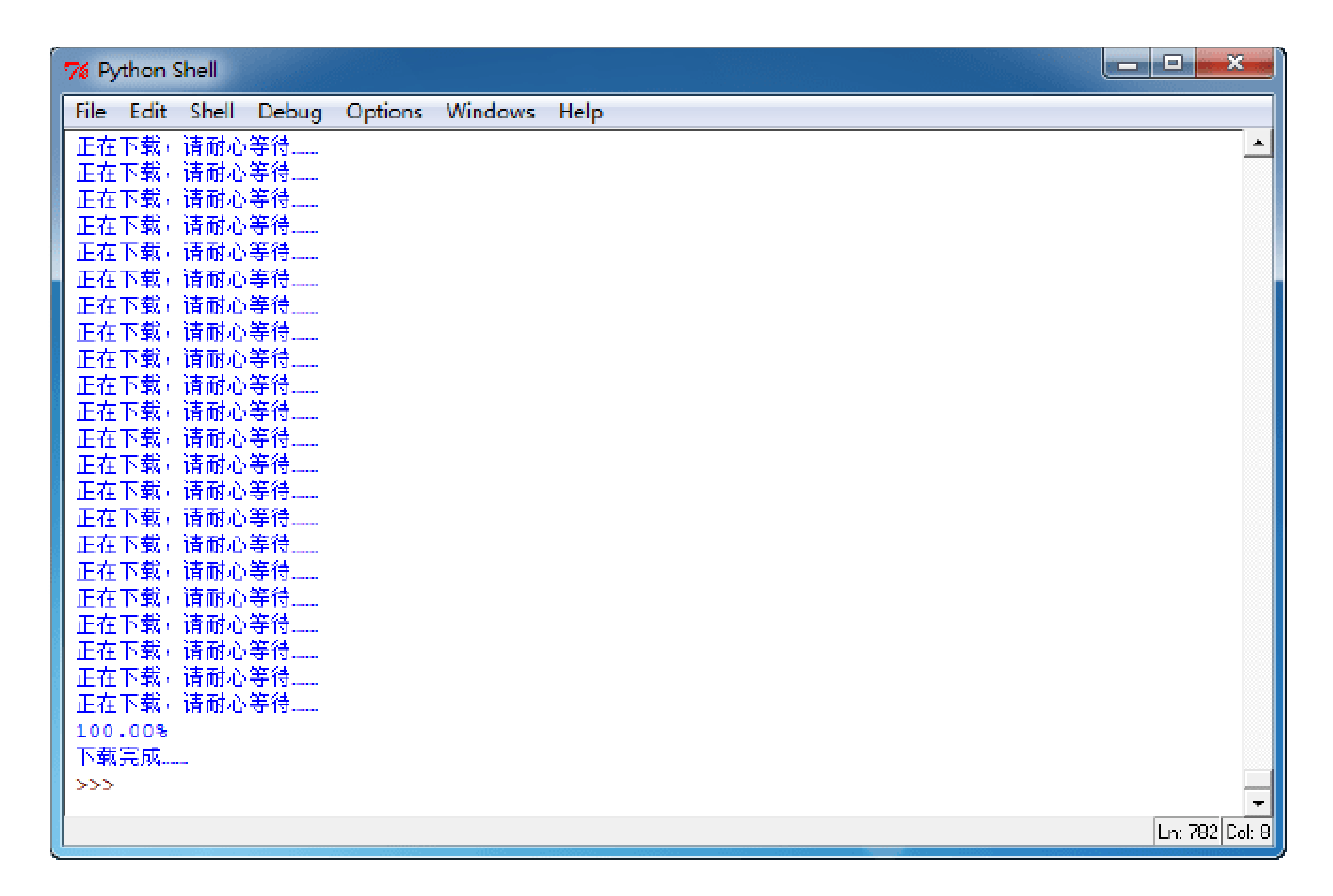

Python Shell
File Edit Shell Debug Options Windows Help
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
正在下载，请耐心等待……
100.00%
下载完成……
>>>
Ln: 782 Col: 8

图 11-17　下载完成



第 12 章　应知应会技能之HTML处理

内容摘要

随着互联网技术的发展和 HTTP 协议的普及，HTML 已经成为一种最热门和最重要的语言之一。只有熟练地掌握了 HTML 语言，才能对现在的各种网页结构有清醒的认识，使客户端和服务器端能够更好地交流。

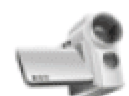
本章将对 Python 标准库中提供的众多处理 HTML 的模块进行详细讲解。例如：对 URL 字符串的处理，获取 HTML 文档的资源，解析 HTML 文档等，最后还将对如何借助 CGI 技术处理客户端数据做进一步的介绍。

学习目标

- 熟练掌握 HTML 的语法规范。
- 掌握对 URL 字符串的处理。
- 了解 CGI 环境的配置。
- 掌握获取 HTML 文档资源的方式。
- 熟练掌握如何解析 HTML 文档。
- 掌握 CGI 的高级应用。

12.1 和我一起回顾HTML

提及 HTML，我想大家都不陌生，也许还会有人自豪地说：我从事美工，我对 HTML 了如指掌，简直到了炉火纯青的地步！。总之，HTML 是我们进入编程世界应该掌握的一门语言，接下来让我们回顾一下 HTML 的发展史以及结构吧。

视频教学：光盘/videos/12/ HTML 概述.avi　　长度：8 分钟

12.1.1 基础知识——HTML概述

HTML(Hyper Text Mark-up Language)即超文本标记语言或超文本链接标示语言，是目前网络上应用最为广泛的语言，也是构成网页文档的主要语言。HTML文本就是由HTML命令组成的描述性文本。HTML的结构包括头部(Head)和主体(Body)两大部分，其中头部描述浏览器所需的信息，而主体则包含所要说明的具体内容。

HTML 最初由 Tim Berners-Lee 在欧洲量子物理实验室创建的，目的是为了使世界各地的物理学家能够方便地进行合作研究。在这个时候 HTML 主要以纯文本格式为基础，可以使用任何文字编辑器处理，随着 HTML 使用率的增加和 Internet 技术的发展，人们对 HTML 规范及标准的需求日益增加。

HTML 由国际组织万维网联盟(W3C)维护，HTML 4.0 加入了很多特定浏览器的元素和属性，同时也将一些浏览器的元素和属性标记为过时。在 Python 中，其模块支持的 HTML 主要以 4.0 版本为基础。

如果你想了解更详细的HTML信息，请访问http://www.w3c.org网站便可了解最新的HTML规范标准。

12.1.2 基础知识——HTML语法规范

当你在浏览器中看到 HTML 中的图片或者不同格式的文本时，是不是很好奇？HTML 文档到底是一个什么工具，功能这么强大？其实 HTML 文档本身只是很普通的文本文件，它可以使用任何文本编辑器来进行编辑。接下来我们写一个简单的 HTML 文件，代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>我是最 easy 的文档</title>
</head>
<body>
<h1 align="center">我家种着万年青 开放每段传奇</h1>
Hello,<b>北京欢迎你</b>
<!--这首歌由众多明星合唱，气势很雄伟啊-->
```

```
<p>你还可以从<a href="http://baike.baidu.com/view/847451.htm">单击这里</a>找到
完整的歌词</p><br />
试试看吧！
</body>
</html>
```

在上述代码中使用了<title>标签，用来显示文档的标题“我是最 easy 的文档”，另外还使用了<h1>、<b>、<p>、
等标签，对这些标签浏览器会做出相应的解释，例如对于<p>标签，在页面上显示了换行。在任一文本编辑器中输入上面的 HTML 源文件，并保存为 easy.html，浏览结果都是相同的，如图 12-1 所示。

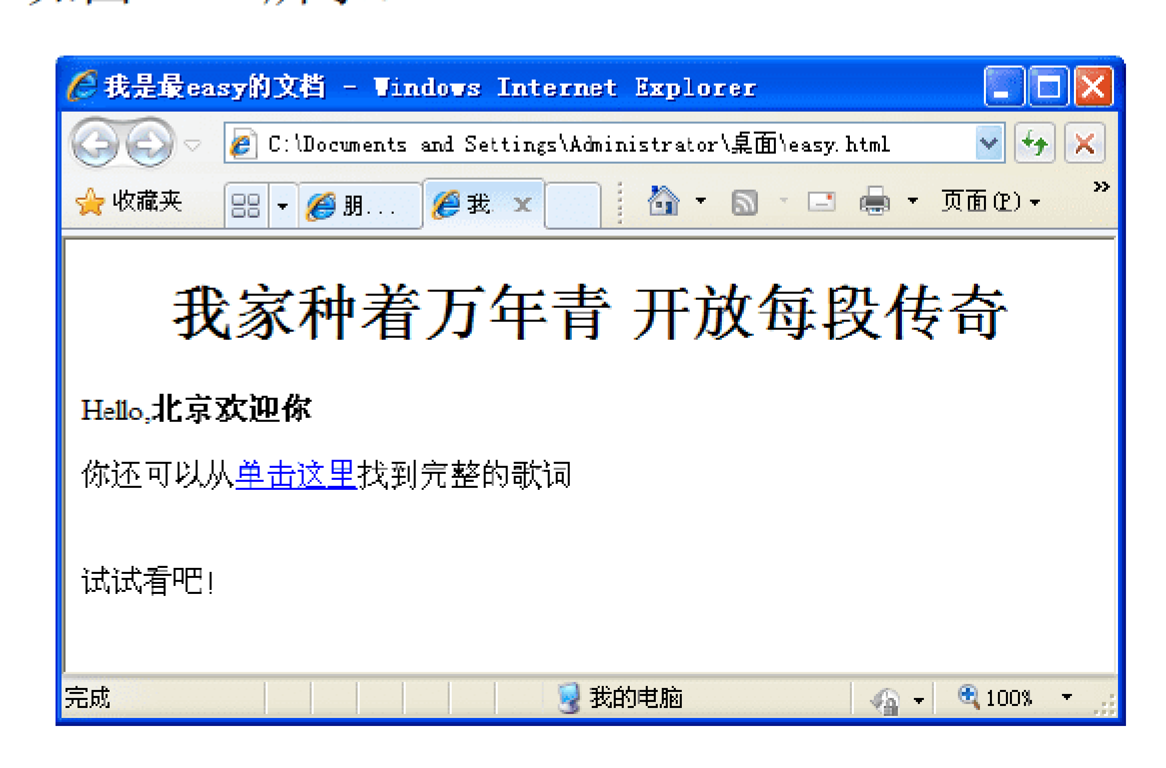


图 12-1　easy.html文件的浏览效果

12.1.3　基础知识——SGML、HTML和XHTML的关系

在上面回顾了 HTML，那么什么是 SGML，什么是 XHTML 呢？它们之间有什么关系？接下来我们将作详细说明。

SGML(Standard Generalized Markup Language，标准通用置标语言)是现时常用的超文本格式的最高层次标准，可以定义置标语言的元语言，其重点主要放在一个文档中的组成部分上，从而使信息的接受者可以不受信息发送者的约束。实际上这是一种通用的文档结构描述标记语言，主要用来定义文献模型的逻辑和物理结构。

一个 SGML 程序由三部分组成，分别是：语法定义、文件类型和文件实例。语法定义中定义的是文件类型和文件实例的语法结构，而在文件类型中的定义的是文件实例的结构和组成结构的元素类型，至于文件实例则是 SGML 语言程序的主体部分。

在最初时期的确没有什么直接的关系，但是随着复杂度的增加，HTML 已经成为 SGML 的一种派生语言。SGML 在实际使用中，每一个特定的 DTD 都定义了一类文件。例如，所有的新闻稿件都可以使用同一个 DTD。由于 HTML 语言是用于万维网的语言，而 SGML 是标记语言的标准，也就是说所有标记语言都是按照 SGML 指定的，因此可以说 HTML 是 SGML 的派生语言之一。

想必大家都知道，HTML 的特点是能够处理结构化的文档结构、字形字体、版面布局、链接等超文本的文档结构。但是你有没有感觉到这样的 HTML 语言规范非常臃肿，内容和格式表现都放在一起，不利于文档分析操作，为了克服这些缺点，XHTML 出现了。

XHTML(eXtensible HyperText Markup Language，可扩展超文本置标语言)是一种置标语言，

表现方式与超文本置标语言(HTML)类似，不过语法上更加严格。在 Python 中处理 HTML 语言，使用 XHTML 将使程序运行更加流畅。

12.2　URL处理

提及 URL，想必大家都知道它是 Internet 上用来定位资源的基础，那么对解析 URL 和获取 URL 资源你了解多少呢？本节将描述 URL 的概念，接着对在 Python 中如何解析和获取 URL 进行详细介绍，同时还将对 Python 中两种不同的模块 urlparse 和 urllib 如何处理 URL 做进一步的研究。

 **视频教学：** 光盘/videos/12/URL 处理.avi　　　　长度：14 分钟

12.2.1　基础知识——统一定位资源URL

统一资源定位符(Universal Resource Locator，URL)是统一资源标识符的一种。统一资源标识符确定一个资源，而统一资源定位符不但确定一个资源，而且还表示出它在哪里。HTML 就是利用该定位符来定位 Internet 上的 HTML 文档信息的，在 Internet 上的每个文档和资源都有唯一一个资源定位符。接下来我们看一下 URL 的语法格式。

```
协议://授权/路径?查询
```

在上述语法中，授权部分一般是服务器的名称或 IP 地址，有时后面还跟一个冒号和一个端口号。它也可以包含接触服务器必需的用户名称和密码。路径部分包含等级结构的路径定义，一般来说不同部分之间以斜线分隔。查询部分一般用来传送对服务器上的数据库进行动态访问时所需要的参数。那么前面提到的协议有哪些呢？表 12-1 列出了比较常用的协议。

表 12-1　比较常用的协议

协议名称	说　明
http	超文本传输协议
https	使用安全套接层的超文本传输协议
ftp	文件传输协议
mailto	电子邮件
file	本地主机上的文件
ldap	轻型目录访问协议
news	Usenet 新闻组
gopher	Gopher 协议
telnet	Telnet 协议

接下来看一下完整的带有授权部分的统一资源标识符的语法。

```
协议://用户名@密码:子域名.域名.顶级域名:端口号/目录/文件名.文件后缀?参数=值#标识
```

从上述语法来看，在用户名、密码、域名和端口号中，域名是最重要的。用户名和密码部分只有在使用 FTP 连接的时候才有可能被用到，因为在使用 FTP 时，大多数连接都使用匿名登录，这时候不需要用户名和密码。端口号只有在 Web 服务器运行在其他非默认端口上时才会被使用。

下面来看一个关于 URL 的小例子。

```
http://baike.baidu.com/view/245485.htm
ftp://ftp.test.com/pub/index.txt
first.html
../images/index.html
```

在这个关于 URL 的小例子中，前面的两个地址都是 URL 语法的简化形式，包含协议、服务器地址等信息。这样的 URL 称为绝对 URL，当浏览器该 URL 时，浏览器能清楚地指导使用什么样的协议和到哪里获取相关的信息。后面两个 URL 不但没有知名访问协议，而且没有给出服务器地址。这样的 URL 称为相对 URL，当浏览器遇到该 URL 时，会将协议默认为 http，从而将服务器地址设置为当前地址。

统一资源定位符一般是区分大小写的，但服务器管理员可以设置大小写是否被区分。有些服务器在收到不同大小写的询问时给出的回复是相同的。在实际操作和网页编写中，最好需要区分大小写。

12.2.2　基础知识——模块urlparse

urlparse 模块主要负责 URL 字符串的解析、拼接等功能。urlparse 模块为操作 URL 字符串提供了 3 种方法，分别是 urlparse()、urlunparse()和 urljoin()。接下来我们来看一下如何使用这些方法，首先来看 urlparse()方法。

1. urlparse()方法

urlparse()方法主要将 URL 字符串拆分成一个 6 元组。

```
scheme://netloc/path;params?query#frag
```

此元组中的每个值都是字符串，在解析 URL 时，将不存在的元组作为一个空字符串，且所有的%转义符都不会被处理。下面通过一个例子来说明，代码如下：

```
import urlparse
url=urlparse.urlparse('http://photo.blog.sina.com.cn/list/blogpic.php?pid=5
230956f')
print url
```

运行程序，执行结果如下：

```
>>>
('http', 'photo.blog.sina.com.cn', '/list/blogpic.php', '', 'pid=5230956f', '')
>>>
```

从运行结果可以看出，urlparse()方法将 URL 地址解析成：协议为 http，服务器地址为 photo.blog.sina.com.cn，路径为/list/blogpic.php，参数为空，查询部分为 pid=5230956f，分片部分为空的 6 元组。

某些时候，你可能只需要获取 URL 地址的某个部分，而非一个元组，该怎么办呢？我们可以通过 urlparse()方法所返回对象的属性来获取，表 12-2 将这些属性列出来了。

表 12-2　urlparse()方法返回对象的属性

属　性	索 引 值	值 含 义	默 认 值
scheme	0	协议	空字符串
netloc	1	服务器地址	空字符串
path	2	路径	空字符串
params	3	参数	空字符串
query	4	查询部分	空字符串
fragment	5	分片部分	空字符串
username		用户名	None
password		密码	None
hostname		主机名	None
port		端口	None



在表 12-2 中，netloc 属性包含了 username、password、hostname 和 port 四个属性的值。

接下来通过一个例子来说明表 12-2 中各个属性的使用，代码如下：

```
import urlparse
url=urlparse.urlparse('http://photo.blog.sina.com.cn/list/blogpic.php?pid=5
230956f')
print 'URL 的协议',url.scheme
print 'URL 的服务器地址',url.netloc
print 'URL 的路径',url.path
print 'URL 的参数',url.params
print 'URL 的查询部分',url.query
print 'URL 的分片部分',url.fragment
print 'URL 的用户名',url.username
print 'URL 的密码',url.password
print 'URL 的主机名',url.hostname
print 'URL 的端口',url.port
```

运行程序，执行结果如下：

```
>>>
URL 的协议 http
URL 的服务器地址 photo.blog.sina.com.cn
URL 的路径 /list/blogpic.php
URL 的参数
URL 的查询部分 pid=5230956f
URL 的分片部分
URL 的用户名 None
URL 的密码 None
URL 的主机名 photo.blog.sina.com.cn
URL 的端口 None
```

```
>>>
```

urlparse()方法还有两个可选的参数 default_scheme 和 allow_fragments。其中，参数 default_scheme 在 URL 中没有提供默认的网络协议或者下载规则时使用，而参数 allow_fragments 则是标识一个 URL 是否使用分片部分，默认值为 True。

2. urlunparse()方法

urlparse 模块的 urlunparse()方法的功能与 urlparse()方法的功能完全相反，urlunparse()方法可以将一个 6 元组拼接成一个 URL 字符串。话不多说，先看一个例子，代码如下：

```
import urlparse
url=urlparse.urlunparse(('http','photo.blog.sina.com.cn','/list/blogpic.php
','','pid=5230956f',''))
print url
```

运行程序，执行结果如下：

```
>>>
http://photo.blog.sina.com.cn/list/blogpic.php?pid=5230956f
>>>
```

如果你想通过为 URL 地址附加新的文件名来处理同一位置下的文件，那么使用 urljoin 方法会是一个不错的选择。

3. urljoin()方法

urlparse 模块中的 urljoin()方法的功能是拼接 URL，其语法格式如下：

```
urljoin(base,url[,allow_fragments])
```

在上述语法中，参数 base 作为其基地址，与第二个参数为相对路径的 url 相结合，组成一个绝对 URL 地址，其中参数 allow_fragments 可根据自己的需要酌情设置。

接下来我们通过一个例子来说明 urljoin()方法的使用，代码如下：

```
import urlparse
print "\n 通过拼接子路径来生成 Python 文档页面的 URL"
newURL =
urlparse.urljoin('http://www.bai.com/admin/',"module-urllib2/request-object
s.html")
XURL =
urlparse.urljoin('http://www.bai.com/admin',"module-urllib2/request-objects
.html")
print 'newURL 的地址是：',newURL
print 'XURL 的地址是：',XURL
```

在上述代码中，首先将 urlparse 模块进行导入，接着使用 urlparse 模块的 urljoin()方法拼接两个 URL 地址 newURL 和 XURL。这两个 URL 地址的不同之处就在于基地址有没有/结尾，那么执行的结果会不会是一样的呢？

```
>>>
通过拼接子路径来生成 Python 文档页面的 URL
newURL 的地址是： http://www.bai.com/admin/module-urllib2/request-objects.html
XURL 的地址是： http://www.bai.com/module-urllib2/request-objects.html
>>>
```

从上述结果可以看出，基地址如果没有以字符/结尾，那么 URL 基地址最右边的部分则会被这个相对路径所替换。

12.2.3 基础知识——URL的编码与解码

urllib 模块提供了一个高级的 Web 交流库，支持 Web 协议、HTTP、FTP 以及 Gopher 协议，同时也支持对本地文件的访问。urllib 模块提供了在给定的 URL 地址下载数据的功能，也可以通过字符串的编码、解码来确保它们是有效 URL 字符串的一部分。在 urllib 模块中提供了 quote()、unquote()、quote_plus()、unquote_plus()和 urlencode()方法，接下来将会对这些方法逐一介绍。

在 URL 中使用的是 ASCII 字符集中的字符，当使用的字符不在 ASCII 字符集中时，就需要对该字符进行编码。ASCII 字符集编码规则是在百分号后面跟着两个十六进制的数字，这和其在 ASCII 字符表中的对应值相同。但有些 ASCII 字符集中的字符却不能使用。比较常见的情况就是在 URL 中使用空格字符，这时空格字符将被编码为%20。这些字符可能会使得 URL 非法，因此被称为保留字符。表 12-3 列出了 URL 编码中保留的字符。

表 12-3 URL编码中保留的字符

保留字符	URL 编码
;	%3B
/	%2F
?	%3F
:	%3A
@	%40
=	%3D
&	%26
空格	%20

在 ASCII 字符集中有些字符会导致上下文歧义，这些字符被称为不安全字符。也可以说是在 URL 中没有特殊的意义，但在 URL 上下文中可能有特殊的意义。例如，双引号在标签中用来分隔属性和属性值，如果在 URL 中出现双引号，那么就有可能使浏览器在解析时发生错误。别急，可以使用%22 来编码双引号，从而避免这种冲突，在表 12-4 中列出了在 URL 编码中出现的不安全字符。

上面我们了解了什么是保留字符和不安全字符，接下来要介绍的是使用 urllib 模块中的方法来对 URL 进行解码和编码，表 12-5 列出了 urllib 模块中比较常用的方法。

表 12-4 URL编码中不安全的字符

不安全字符	URL 编码
<	%3C
>	%3E

"	%22
#	%23
%	%25
{	%7B
}	%7D
|	%7C
\	%5C
^	%5E
~	%7E
[	%5B
]	%5D
'	%60

表 12-5　urllib模块中URL编码和解码的方法

方　法	功　能
quote(s[, safe = '/'])	返回一个所有特殊字符(这些字符在 URL 中有特殊含义)都被对 URL 友好的字符串代替的字符串(例如%7E 代替了~)，如果需要在 URL 中使用一个包含特殊字符的字符串，这个函数就很有用。safe 字符串默认的是/
quote_plus(s[, safe=' '])	该函数的功能和 quote()函数的功能差不多，只是把 s 中的空格使用+来代替，返回转换后的字符串
unquote(s)	该函数的功能和 quote()函数的功能相反
unquote_plus(s)	该函数的功能和 quote_plus()函数的功能相反

下面通过例子来说明表 12-5 中 urllib 模块中方法的使用，其代码如下：

```
import urllib
name='admin'
num=6
base='http://www.baidu.com /~dcy'
final='%s?name=%s&num=%d'%(base,name,num)
print '没有使用 quote 方法：',final
print '使用 quote 方法：',urllib.quote(final)
print '使用 quote_plus 方法：',urllib.quote_plus(final)
```

在上述代码中，首先导入 urllib 模块，接着将声明的 name、num 和 base 值赋给 final，分别使用方法 quote 和 quote_plus 对地址 final 进行编码，其执行结果如下：

```
>>>
没有使用 quote 方法： http://www.baidu.com /~dcy?name=admin&num=6
使用 quote 方法： http%3A//www.baidu.com%20/%7Edcy%3Fname%3Dadmin%26num%3D6
使用 quote_plus 方法：
http%3A%2F%2Fwww.baidu.com+%2F%7Edcy%3Fname%3Dadmin%26num%3D
6
>>>
```

从上述显示结果你有没有看出什么端倪？原来在使用 quote 方法将该 URL 进行编码时，地址中的空格被编码成%20；使用 quote_plus 方法进行编码时，地址中的空格则会被编码成+进行连接。

当你在百度搜索栏上输入你所查询的内容并敲回车键时，地址栏中会有一大堆你看不懂的“乱码”，你是不是很好奇，服务器如何辨别查询的内容呢？别急，只需使用 urllib 模块中的方法将 URL 解码就能看到你输入的内容了，请看 unquote 方法的使用。

```
import urllib
url="http://www.baidu.com/s?+bs=%CA%AE%CD%F2%B8%F6%CE%AA%CA%B2%C3%B4%D6%D0%
CE%AA%CA%B2%C3%B4%C3%BB%D3%D0%BF%BC%CA%D4%B2%BB%BC%B0%B8%F1"
print '使用 unquote 方法：',urllib.unquote(url)
print '使用 unquote_plus 方法：',urllib.unquote_plus(url)
```

在上述代码中，url 是在百度地址栏上显示的字符串，分别使用 unquote 和 unquote_plus 方法将 url 解码，其结果如下：

```
>>>
使用 unquote 方法： http://www.baidu.com/s?+bs=十万个为什么中为什么没有考试不及格
使用 unquote_plus 方法： http://www.baidu.com/s? bs=十万个为什么中为什么没有考试不及格
>>>
```

从执行结果可以看出，unquote 和 quote 方法相反，只是一个将 URL 解码，另外一个将 URL 编码而已。同样，unquote_plus 和 quote_plus 方法也是一个将+符号解码成空格，一个将空格编码成+符号。

12.3 CGI：帮助Web服务器处理客户端数据

CGI 即(Common Gateway Interface，公共网关接口)是外部应用程序和 HTTP 服务器之间交互的一个通用接口标准。CGI 既不是一种语言，也不是一种网络协议，它定义的是 HTTP 服务器和程序之间交换信息的规范，因此 CGI 可以使用任何语言来编写，包括 Python 语言。

视频教学：光盘/videos/12/GML.avi　　　　**长度：**9 分钟

12.3.1 基础知识——CGI介绍

CGI 是一种动态网页构建技术，虽然需要由服务器提供，但和服务器却是独立的。只有 HTTP 服务器支持 CGI，便可运行相应的脚本。CGI 脚本文件通过环境变量、命令行参数和标准输入/输出流与 HTTP 服务器进行通信，传递参数并对该参数进行处理。当传递的方式为 GET 时，CGI 程序通过环境变量来获取客户端提交的数据；当传递的方式为 POST 时，CGI 程序将通过标准输入流和环境变量来获取客户端提交的数据。当 CGI 程序返回处理结果给客户端时，则是通过标准输出流将数据输出到服务器进程中。

当客户端请求一个 CGI 程序时，CGI 需要输出信息来声明请求的 MIME 类型，并通过服务器来传递到客户端。CGI 的环境变量 HTTP_ACCEPT 提供了可以被客户端和服务器端接受

的 MIME 类型列表。表 12-6 列出了 HTTP 所支持的 MIME 类型。

表 12-6　HTTP支持的MIME类型

类　型	子 类 型	说　明
Application	Octet-stream	MIME 编码的二进制数据
Audio	Basic	音频文件
Image	GIF、JPEG	图像文件
Message	External-body	对于信息的封装
Multipart	Mixed digest	混合类型，可以支持文件上传等操作
Text	HTML、Plain	HTML 文档，纯文本文档
Video	MPEG	视频文件

在这里，我们主要介绍的 MIME 类型是 text/html，也就是大家经常见到的 HTML 文档。

CGI 脚本输出主要包括两部分，分别是：文件头和具体的文件信息。首先来看文件头信息在 Python 中如何表示。

文件头部分主要表示的是 MIME 类型的信息，例如在 Python 中输出 MIME 类型的 text/html 文件时，使用的代码格式如下：

```
print "Content-Type:text/html"
print
```

在上述代码格式中，第一行代码主要指的是下面要输出的内容为 HTML 文档。第二行代码指的是打印一个空行，表示文件头的结束。当然也可以将这两行代码合并在一起，格式如下：

```
print "Content-Type:text/html\n\n"
```

你是不是很奇怪，为什么要输入两个换行符呢？这是因为在这个部分的后面需要输出一个完整的空行来表示文件头的结束。

文件具体的信息，在 Python 中编写如下代码：

```
print '<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">'
print '<html xmlns="http://www.w3.org/1999/xhtml">'
print '<head>'
print '</head>'
print '<body>'
#省略部分代码
print '</body>'
print '</html>'
```

将上面介绍的文件头和文件具体的信息结合成一个文件，并保存到 CGI 脚本目录下。当客户端请求该脚本文件时，系统将返回上述的 HTML 文档。

在 Python 标准库中的 cgi 模块可以用来处理 CGI 脚本，该模块提供了丰富的方法来对各种信息进行输入和输出。如何运行脚本成了我们最头疼的难题，下面将对此作详细介绍。

12.3.2 基础知识——配置和获取CGI环境

运行时，由于每个 CGI 脚本都会调用外部程序来生成 HTTP 输出，当请求比较多的时候将会使服务器的负载过重，导致性能降低。另外就是不安全，因为通过 CGI 脚本可以直接调用系统中的程序，并执行相应的操作和访问系统文件，这样系统的漏洞可能会通过 CGI 脚本的执行而暴露出来。

这么说 CGI 是不是有很多弊端？如何解决呢？别急，我们可以通过 mod_python 模块来提高 Apache 服务器对 Python 脚本文件的响应速度。至于安全方面，现在的 HTTP 服务器将 CGI 脚本文件限制在一个特定的文件中，如 cgi-bin 文件。这样就阻止了在不经意的情况下将 CGI 脚本暴露给外界，而只有通过确认的 CGI 脚本文件方可放到特定的文件中，这样就很安全了。

接下来我们看一下如何访问 CGI 脚本。

(1) 从 http://download.csdn.net/source/3114579 下载 Apache 服务器，版本是 Apache 2.2。

(2) 双击 Apache HTTP Server 2.2 进行安装，如图 12-2 所示。

(3) 单击 Next 按钮，选择接受许可单选按钮，如图 12-3 所示。

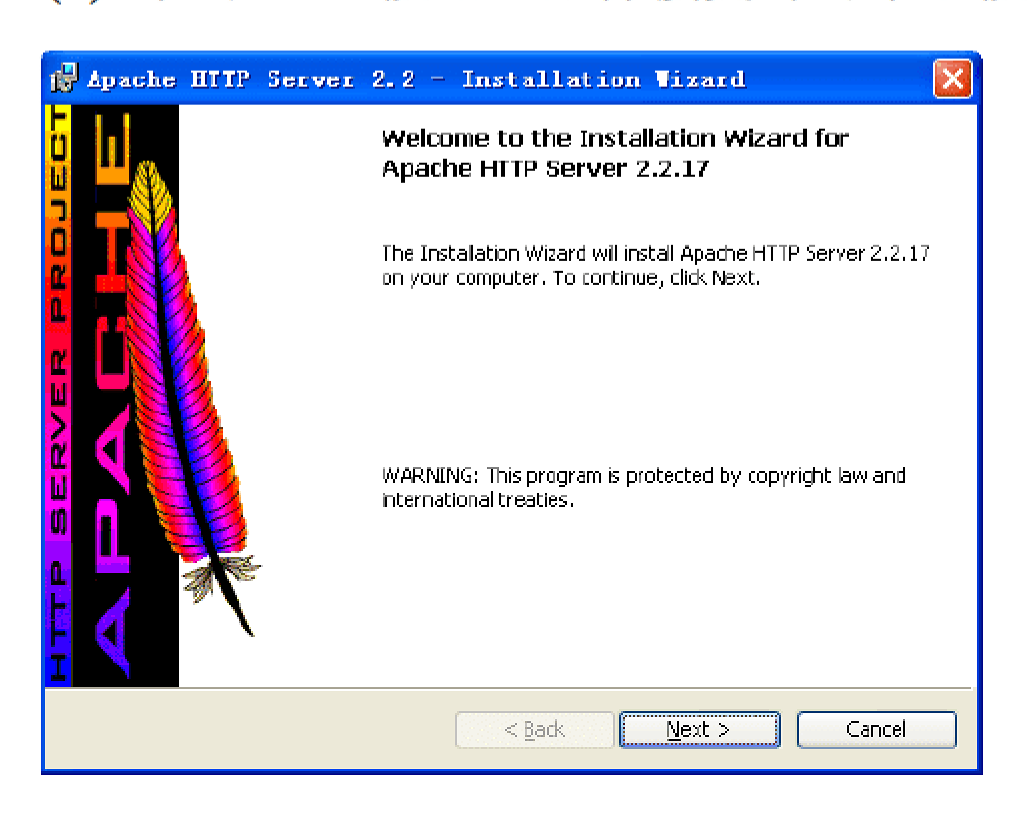


图 12-2 Apache安装向导

图 12-3 接受安装许可

(4) 单击 Next 按钮，分别在名称为 Network Domain 文本框和名称为 Server Name 文本框中填写 www.itzcn.com，在 Administroator's Email Address 文本框中填写 itzcn@itzcn.com，如图 12-4 所示。

(5) 单击 Next 按钮，将安装地址设置为 D:\Apache2.2\，之后单击 Install 按钮，最后单击 Finish 完成。

(6) 在地址 http://download.csdn.net/down/1782221/xiao80xiao 上下载 mod_python-3.3.1，然后双击进行安装，如图 12-5 所示。

(7) 单击“下一步”按钮，选中 Python Version 2.5，如图 12-6 所示。



图 12-4　填写服务器配置信息

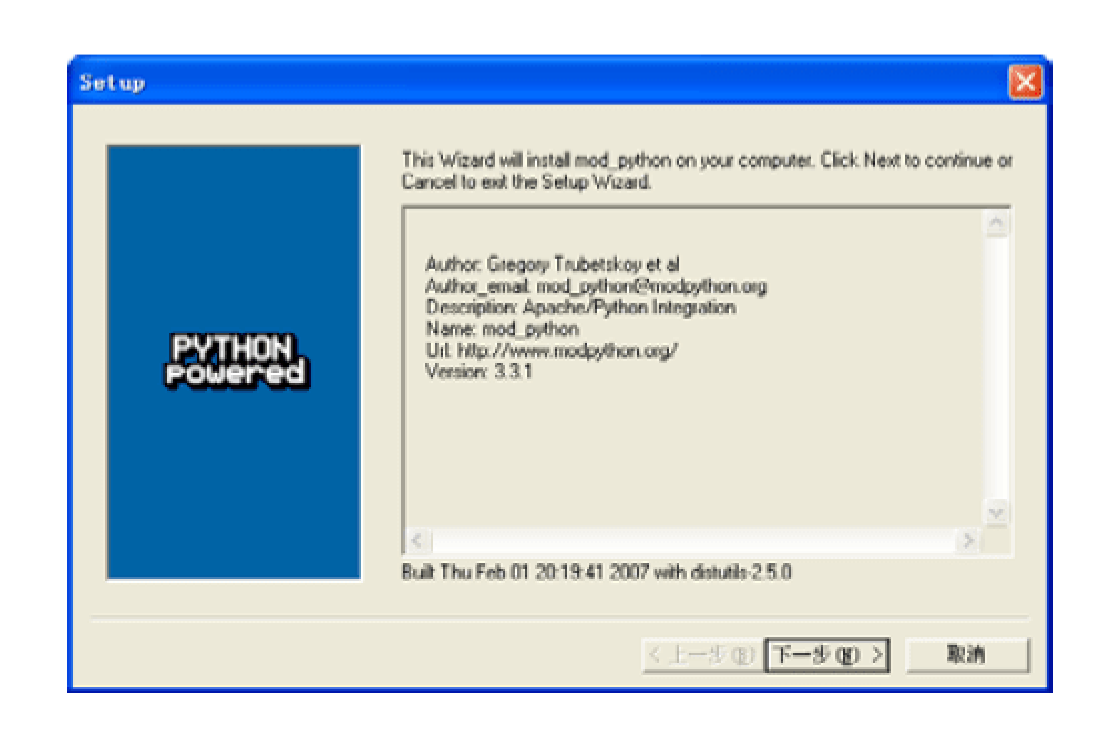


图 12-5　mod_python安装向导

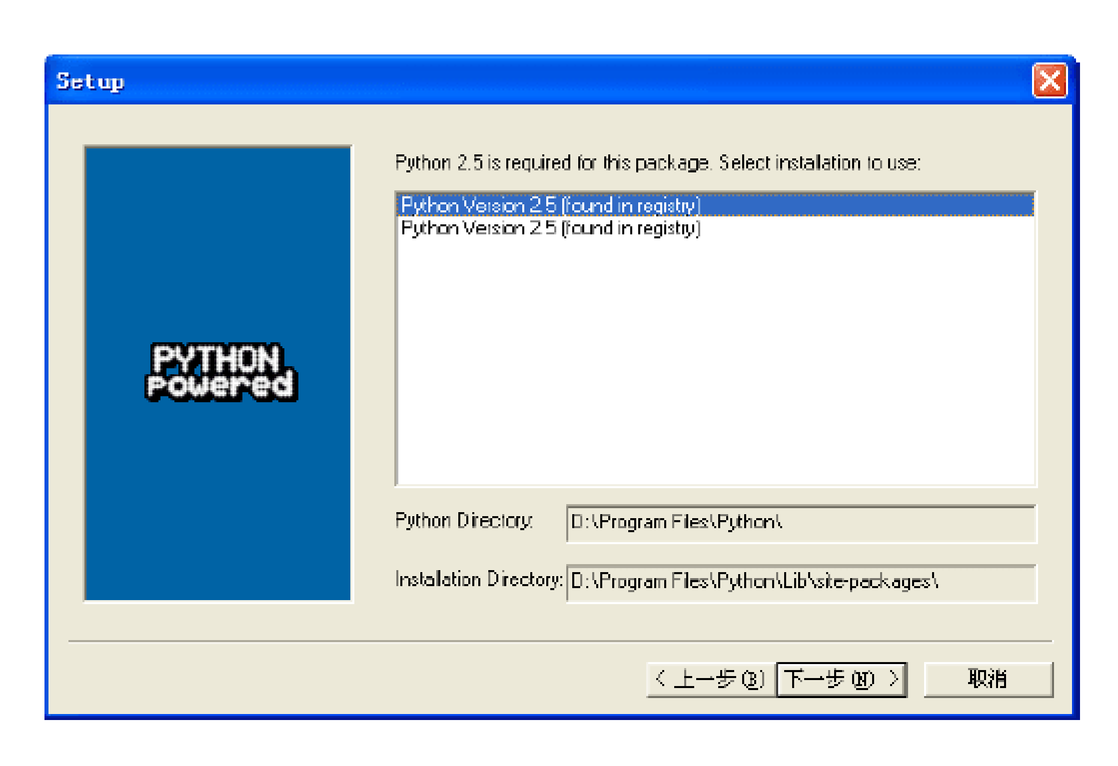
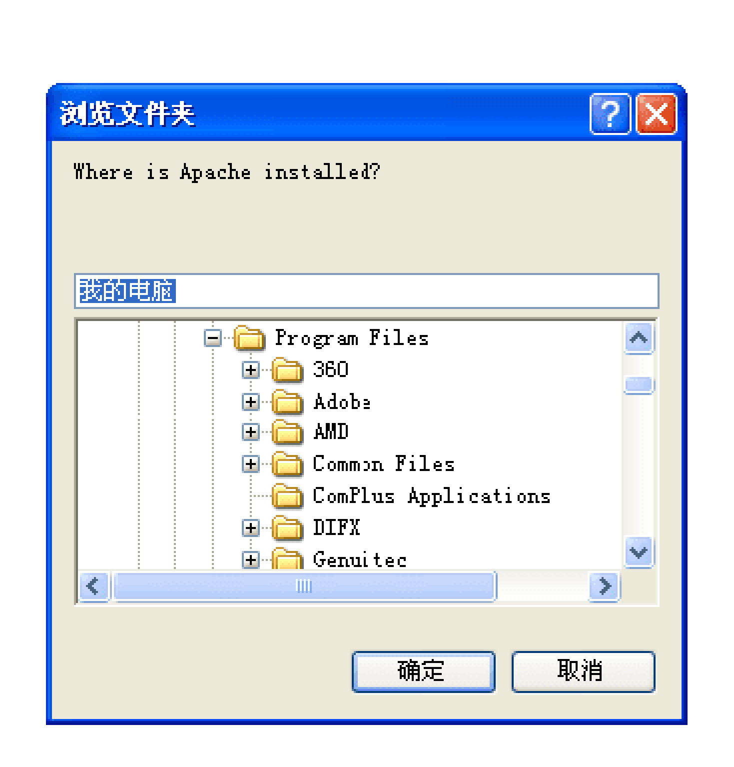


图 12-6　选中安装Python Version 2.5

(8) 一直单击“下一步”按钮，直到出现“浏览文件夹”对话框，如图 12-7 所示。

(9) 这里，选中 Apache 2.2 的安装路径 D:\Apache2.2\，单击“下一步”按钮，如图 12-8 所示。



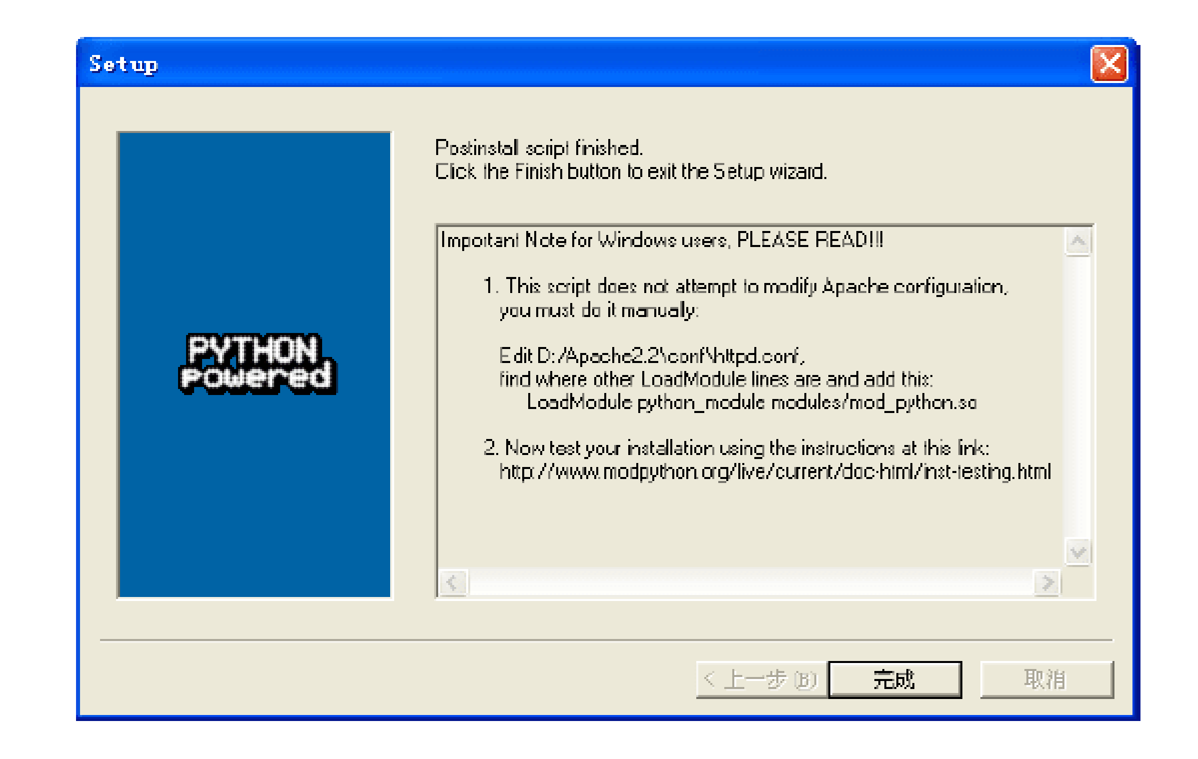


图 12-7　浏览文件夹

图 12-8　选中路径后

(10) 单击“完成”按钮，即安装成功。

(11) 打开 D:\Apache2.2\conf 文件中的 httpd.conf 文件，找到 LoadModule，然后添加代码如下：

```
LoadModule python_module modules/mod_python.so
```

(12) 找到<Directory>标记，修改代码如下：

```
<Directory D:/Apache2.2/cgi-bin/>
   AddHandler mod_python .py
   PythonDebug On
</Directory>
```

接下来通过一个例子来说明，其代码如下：

```
#!D:/Program Files/Python/python.exe
print "Content-Type:text/html\n\n"
import datetime
print datetime.datetime.now()
```

在上述代码中，第一行代码指定 Python 可执行文件的位置。之后将 datetime 模块导入，使之打印当前的时间。最后保存文件，名称为 mydate.cgi，保存地址为 D:\Apache2.2\cgi-bin。

(13) 打开 D:\Apache2.2\conf 文件中的 httpd.conf 文件，在<Directory>标记中添加代码如下：

```
PythonHandler mydate
```

(14) 保存修改好的代码。在浏览器地址栏中输入地址 http://localhost/cgi-bin/mydate.cgi，访问结果如图 12-9 所示。

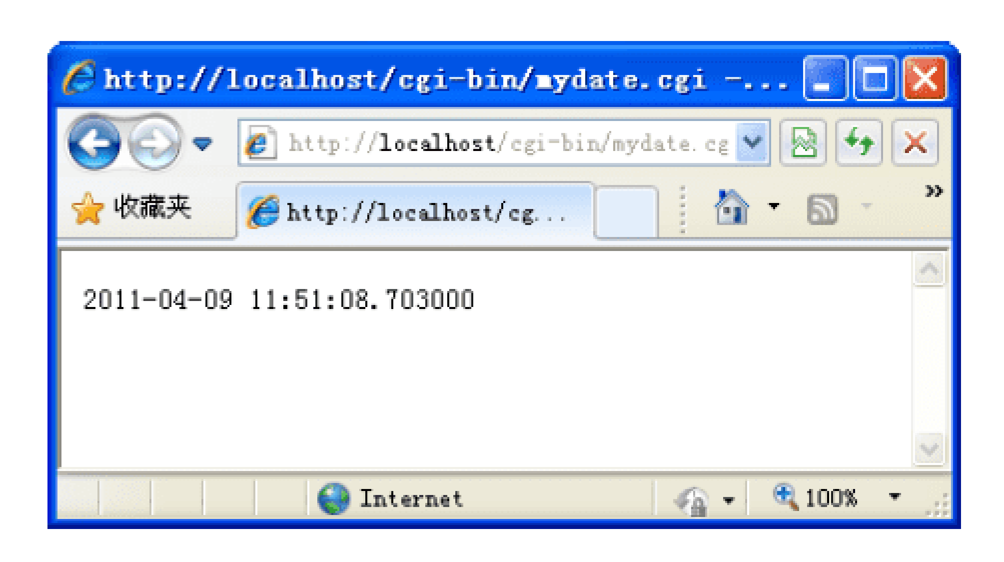


图 12-9　显示当前时间

这样 CGI 环境准备完毕。这里是否存在环境变量呢？答案是肯定的。可以通过这些变量对请求做一些简单的操作。例如获得客户端的 IP 地址，下面就来看一下如何使用环境变量 REMOTE_ADDR 获得客户端 IP 的例子，代码如下：

```
#!D:/Program Files/Python/python.exe
print "Content-Type:text/html\n\n"
import os
remoteAddr=os.environ['REMOTE_ADDR']
if remoteAddr=='127.0.0.1':
   print '我是来自本地的访问',remoteAddr
else:
   print '我是来自外部的访问'
```

在上述代码中，首先将 os 模块导入，接着使用 os 的 environ 方法获取环境变量 REMOTE_ADDR 的值，最后进行判断并打印输出。



打开 D:\Apache2.2\conf 文件中的 httpd.conf 文件，在<Directory>标记中添加代码 PythonHandler remoteAddr，否则会出现服务器错误。

在浏览器地址栏中输入 http://localhost/cgi-bin/remoteAddr.cgi，执行结果如图 12-10 所示。

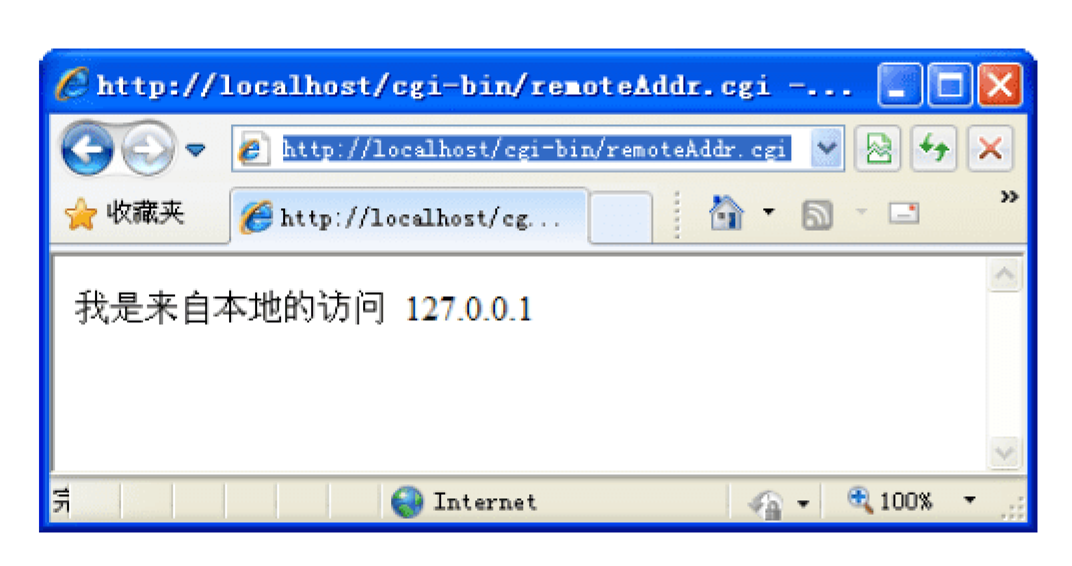


图 12-10　运行脚本remoteAddr.cgi

前面介绍了环境变量 REMOTE_ADDR 的使用，那么还有没有其他的变量呢？别急，cgi 模块为我们提供了 print_environ 函数，通过使用 print_environ 可以得到 CGI 环境中所有可用的变量信息。

创建一个.py 文件，并添加代码如下：

```
import cgi
print cgi.print_environ()
```

执行结果如下：

```
>>>
<H3>Shell Environment:</H3>
<DL>
<DT> ALLUSERSPROFILE <DD> C:\Documents and Settings\All Users
<DT> APPDATA <DD> C:\Documents and Settings\Administrator\Application Data
<DT> CLIENTNAME <DD> Console
<DT> COMMONPROGRAMFILES <DD> C:\Program Files\Common Files
<DT> COMPUTERNAME <DD> PC2011010514TLL
<DT> COMSPEC <DD> C:\WINDOWS\system32\cmd.exe
<DT> FP_NO_HOST_CHECK <DD> NO
<DT> HOME <DD> C:\Documents and Settings\Administrator
<DT> HOMEDRIVE <DD> C:
<DT> HOMEPATH <DD> \Documents and Settings\Administrator
<DT> LOGONSERVER <DD> \\PC2011010514TLL
<DT> NUMBER_OF_PROCESSORS <DD> 2
<DT> OS <DD> Windows_NT
<DT> PATH <DD> D:\Program
Files\Python\Lib\site-packages\Pythonwin\;D:\Program
Files\Python\Lib\site-packages\Pythonwin\;d:\Ruby192\bin;C:\WINDOWS\system3
2;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\Common
Files\Adobe\AGL;D:\MySQL\MySQL Server 5.1\bin;C:\Program Files\Common
Files\Thunder Network\KanKan\Codecs;C:\Program Files\Microsoft SQL
Server\100\Tools\Binn\;C:\Program Files\Microsoft SQL
Server\100\DTS\Binn\;C:\WINDOWS\system32\WindowsPowerShell\v1.0;C:\Program
Files\Microsoft SQL Server\100\Tools\Binn\VSShell\Common7\IDE\
<DT> PATHEXT
<DD> .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.RB;.RBW;.PSC1;.py;.pyw
<DT> PROCESSOR_ARCHITECTURE <DD> x86
<DT> PROCESSOR_IDENTIFIER <DD> x86 Family 15 Model 107 Stepping 1, AuthenticAMD
<DT> PROCESSOR_LEVEL <DD> 15
<DT> PROCESSOR_REVISION <DD> 6b01
<DT> PROGRAMFILES <DD> C:\Program Files
<DT> SESSIONNAME <DD> Console
<DT> SYSTEMDRIVE <DD> C:
<DT> SYSTEMROOT <DD> C:\WINDOWS
<DT> TCL_LIBRARY <DD> D:\Program Files\Python\tcl\tcl8.4
<DT> TEMP <DD> C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp
```

```
<DT> TIX_LIBRARY <DD> D:\Program Files\Python\tcl\tix8.4
<DT> TK_LIBRARY <DD> D:\Program Files\Python\tcl\tk8.4
<DT> TMP <DD> C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp
<DT> USERDOMAIN <DD> PC2011010514TLL
<DT> USERNAME <DD> Administrator
<DT> USERPROFILE <DD> C:\Documents and Settings\Administrator
<DT> VS100COMNTOOLS <DD> C:\Program Files\Microsoft Visual Studio
10.0\Common7\Tools\
<DT> WINDIR <DD> C:\WINDOWS
</DL>
None
>>>
```

从上述结果可以看出，第一个元素是环境变量，而第二个元素则是该环境变量的值信息，看，这样当前服务器支持的 CGI 变量就一目了然了。

为了让 CGI 和客户端能够更好地进行交互，表 12-7 列出了 CGI 环境中重要的环境变量。

表 12-7　CGI中的部分环境变量

环境变量	说　明
REMOTE_ADDR	服务器的地址
PATH_INFO	路径信息
REQUEST_METHOD	客户端请求的方法，GET 或者 POST
SERVER_NAME	服务器名字
SERVER_PORT	服务器端口
SCRIPT_NAME	服务器执行脚本的名字
HTTP_USER_AGENT	客户端软件
SERVER_PROTOCOL	服务器支持的协议
SERVER_SOFTWARE	服务器端的软件信息
QUERY_STRING	查询字符串

12.3.3　基础知识——解析用户的输入

CGI 技术的一个重要作用就是可以获取用户的输入。你可能会疑惑请求的方式有两种(POST 和 GET)，它们获取用户输入的方式是否一样，会不会很麻烦呢？不用担心，cgi 模块使用了一种统一的方式来处理用户的输入，即 FieldStorage 类。

FieldStorage 类的使用方法很简单，只需要简单地初始化，可以不含参数。也可以将 FieldStorage 类的对象看成 Python 语言中的字典结构，同样也支持字典访问的部分方法。

接下来通过一个例子来看一下 FieldStorage 类的使用，代码如下：

```
#!D:/Program Files/Python/python.exe
print "Content-Type:text/html\n\n"
import cgi
form=cgi.FieldStorage()
for key in form.keys():
    print key,'value:',form[key].value
```

```
    print '<br/>'
```

在上述代码中，首先将 cgi 模块导入，接着实例化一个没有参数的 FieldStorage 类对象 fs，然后遍历 fs 对象中的所有键和值并显示相应的信息，最后通过
方式实现换行。

打开浏览器，输入地址 http://localhost/cgi-bin/remoteAddr.cgi?username=duanchunyang&userpass=lintiantian，显示效果如图 12-11 和图 12-12 所示。

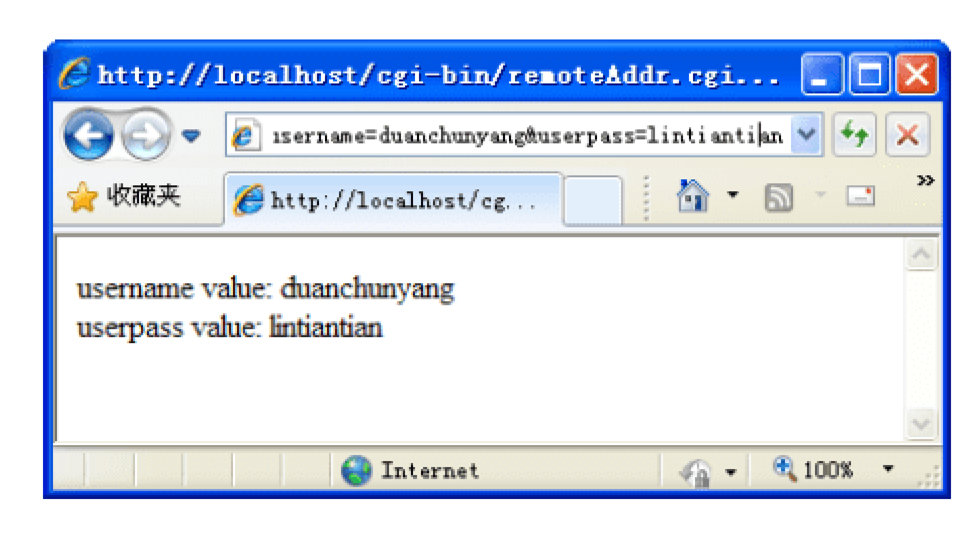




图 12-11　接收参数显示的效果

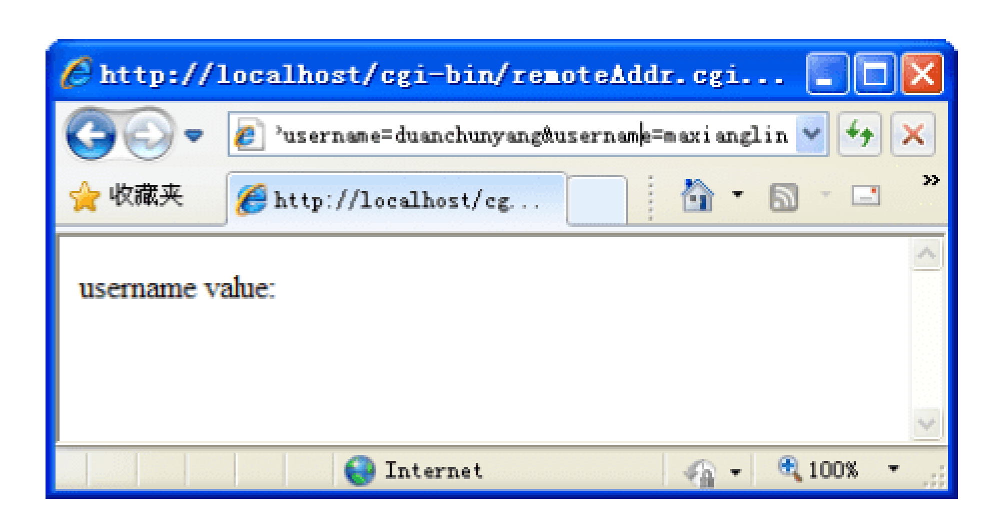


图 12-12　两个参数相同时的效果

从图 12-11 可以看出，在地址栏上传入的参数为 username=duanchunyang&userpass=lintiantian，因此可以断定使用的提交方式是 GET。

但是如果在地址栏上传入的参数有两个 username 值，执行结果则如图 12-12 所示。很显然那不是我们想要的效果，想要解决这个问题，需要使用 FieldStorage 类的 getlist 方法，这样就可以获取一个字段的多个值了。将上述代码修改如下所示：

```
#!D:/Program Files/Python/python.exe
print "Content-Type:text/html\n\n"
import cgi
fs=cgi.FieldStorage()
for key in fs.keys():
    print key,'value:',fs.getlist(key)
    print '<br/>'
```

在上述代码中，在循环中使用了 getlist 方法来获取对应字段的值。再看一下执行结果，如图 12-13 所示。

从图 12-13 可以看出，使用 FieldStorage 类的 getlist 方法将一个字段的多个值接收并保存到一个列表中，这样用户便可以针对该列表做进一步的处理。

在上面处理用户输入的过程中，并没涉及特殊字符。用户可能输入的是 HTML 标签一类的字符，那么就需要将该字符进行转义，否则显示结果将会出现偏差，其结果如图 12-14 和图 12-15 所示。

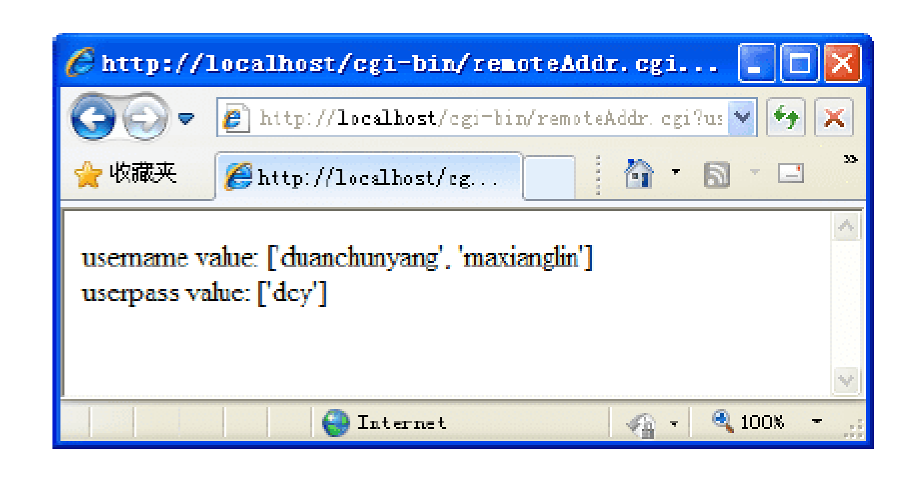


图 12-13　获取一个字段的多个值的效果

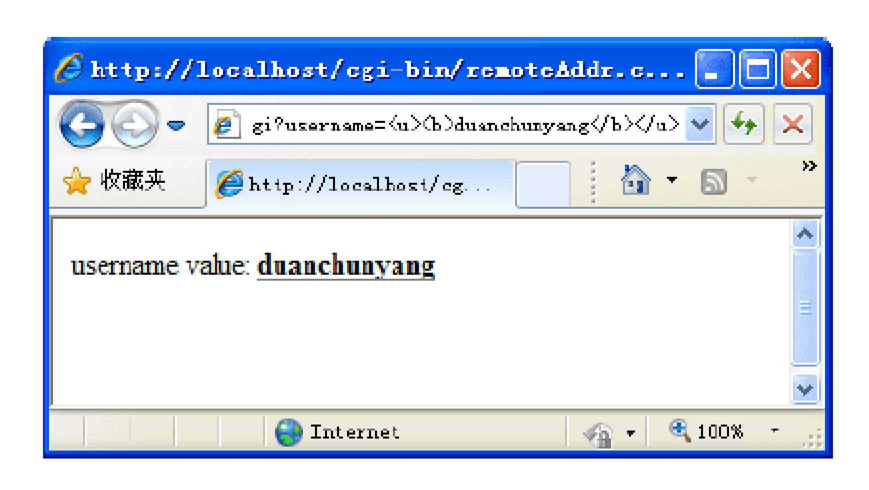



图 12-14　没有进行转义的效果

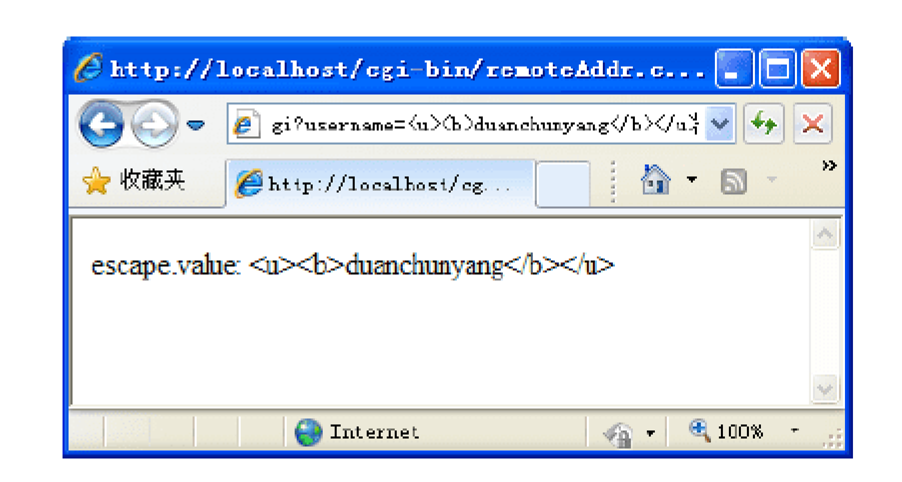


图 12-15　进行转义之后的效果

从图 12-14 中可以看出，并没有将 username 直接输出，由于参数 username 中包含 HTML 标签，因此对这些标签进行了解析，所以显示结果是 username 的值加粗并且有下划线。那么如何解决这样的问题呢？别急，cgi 模块提供了 escape 方法，使用该方法可以将&、<、>以及 HTML 标签等字符进行转义，从而使接收的值可以正常显示。代码修改如下：

```
#!D:/Program Files/Python/python.exe
print "Content-Type:text/html\n\n"
import cgi
fs=cgi.FieldStorage()
for key in fs.keys():
    print 'escape.value:',cgi.escape(fs[key].value)
    print '<br/><br/>'
```

执行结果如图 12-15 所示。

12.4　获取HTML文档资源

在 Python 中获取 HTML 资源的方式有多种，其中在 urllib 模块中获取网络资源的方式有两种，分别是 urlopen 和 urlretrieve。另外还可以使用 urllib2 或者 httplib 模块来获取 HTML 资源。下面将对这些模块以及模块中的方法进行详细介绍，首先来看如何使用 urllib 模块中的方法获取 HTML 资源。

**视频教学：**光盘/videos/12/获取 HTML 文档资源.avi　　**长度：**13 分钟

12.4.1　基础知识——使用urlopen方法获取HTTP资源

在第 11 章中已经讲解过 urllib2 模块中的 urlopen()方法，其实在 urllib 模块中也有一个 urlopen()方法，该方法与 urllib2 模块中的 urlopen()方法功能相同，不仅可以打开一个远程文件，也能获取 HTTP 资源。

使用 urlopen()方法返回的是一个文件类型的对象，如果该文件对象的名称为 fabs，那么该对象 fabs 提供的方法如表 12-8 所示。

表 12-8　urlopen()返回对象的常用方法

方　法	说　明
fabs.read()	从 fabs 中读取所有内容(字节)
fabs.readline()	从 fabs 中读取一行
fabs.readlines()	从 fabs 中读取所有行并放回一个列表
fabs.close()	关闭 fabs 的 URL 连接
fabs.fileno()	返回 fabs 的文件句柄
fabs.info()	获得 fabs 的 MIME 头文件
fabs.geturl()	返回 fabs 所打开的真正 URL

下面通过一个例子来说明表 12-8 中方法的使用。

(1) 我们创建一个名称为 ceshi.txt 文件，其内容如下：

```
你是我心内的一首歌
心间开启花一朵
你是我生命的一首歌
想念汇成一条河
点在我心内的一首歌
```

(2) 创建一个名称为 file.py 的文件，并添加代码如下：

```
import urllib
fabs = urllib.urlopen("a.txt")
print '获取文件中的所有信息：',fabs.info()
print '只读一行数据:',fabs.readline()
print '从该文件对象中读取 bytes 字节 ',fabs.read()
print '获取文件句柄:',fabs.fileno()
print '打开真正的 url:',fabs.geturl()
fabs.close()
```

在上述代码中，首先将 urllib 模块导入，接着使用 urlopen()方法返回一个文件对象 fabs，然后使用该对象的方法获取所需要的信息，最后使用 close()方法将文件对象关闭。其执行结果如下：

```
>>>
获取文件中的所有信息： Content-Type: text/plain
Content-Length: 92
Last-modified: Wed, 06 Apr 2011 08:47:26 GMT
只读一行数据：你是我心内的一首歌
从该文件对象中读取 bytes 字节  心间开启花一朵
你是我生命的一首歌
想念汇成一条河
点在我心内的一首歌
获取文件句柄： 3
打开真正的 url: a.txt
>>>
```

12.4.2 基础知识——使用httplib模块获取资源

httplib 模块实现了 HTTP 和 HTTPS 协议的客户端，该模块主要在访问 HTTP 资源的时候使用，一般情况下该模块主要作为 urllib 的基础模块，并不直接使用。

httplib 模块提供了 HTTPConnection、HTTPSConnection 和 HTTPResponse 类来处理具体的 URL 资源。其中 HTTPConnection 类提供的构造函数所需要的参数为 HTTP 服务器，默认端口为 80。下面看一下 HTTPConnection 实例对象的构造方式。

```
connection1=httplib.HTTPConnection("www.baidu.com")
connection2=httplib.HTTPConnection("www.baidu.com:80")
connection3=httplib.HTTPConnection("www.baidu.com",80)
```

上面使用了 3 种构造方式，其中 connection1 使用的是默认构造方式，其端口为 80；connection2 使用冒号来分隔 URL 地址和端口；而 connection3 使用了第二个参数 port，并将该 80 端口值赋给该参数。

上面使用 3 种方式生成 3 个连接对象，接着可以使用该连接对象的方法来获取相对应的资源。其方法主要有 request，该方法接收两个参数，分别是访问的方式 GET 或者 POST 和 URL 资源。下面来看一下这 3 种构造方式的使用，首先来看第一种构造方式，代码如下：

```
import httplib
connection=httplib.HTTPConnection('www.python.org')
connection.request('GET','/index.html')
re=connection.getresponse()
print '--------connection------------'
print re.status,re.reason
print re.read(256)
connection.close()
```

在上述代码中，首先将 httplib 模块导入，接着使用 HTTPConnection 方法返回一个连接对象，该 HTTP 的服务器端口为默认端口，然后调用 request 方法来获取指定服务器的 index.html 文档资源，之后使用 getresponse 方法获得一个 HTTPResponse 实例对象 re，最后使用 re 的 status 和 reason 属性，分别获取服务器返回的状态码和服务器返回状态的原因。执行结果如下：

```
>>>
--------connection------------
200 OK
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <meta http-equiv="content-type" content="text/html; ch
>>>
```

下面来看第二种构造方式的使用，代码如下：

```
import httplib
connection1=httplib.HTTPConnection('//www.baidu.com:8080')
connection1.request('POST','/index.html')
re1=connection1.getresponse()
print '------------connection1-------------'
print re1.status,re1.reason
data1=re1.read()
print data1
```

```
connection1.close()
```

在上述代码中，在方法 HTTPConnection 中传入的参数不是默认的服务器端口，而是以使用冒号将 URL 地址和端口分隔的方式为参数，接着使用 request 方法获取文档资源，然后使用 getresponse 方法返回一个实例对象 re1，最后使用该实例对象的 status 和 reason 属性来获取服务器的状态码等信息。执行结果如下：

```
>>>
------------connection1-------------
404 /index.html
<html><head><title>Apache Tomcat/5.5.23 - Error report</title><style><!--H1
{font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;f
ont-size:22px;} H2
{font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;f
ont-size:16px;} H3
{font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;f
ont-size:14px;} BODY
{font-family:Tahoma,Arial,sans-serif;color:black;background-color:white;} B
{font-family:Tahoma,Arial,sans-serif;color:white;background-color:#525D76;} P
{font-family:Tahoma,Arial,sans-serif;background:white;color:black;font-size
:12px;}A {color : black;}A.name {color : black;}HR {color : #525D76;}--></style>
</head><body><h1>HTTP Status 404 - /index.html</h1><HR size="1"
noshade="noshade"><p><b>type</b> Status report</p><p><b>message</b>
<u>/index.html</u></p><p><b>description</b> <u>The requested resource
(/index.html) is not available.</u></p><HR size="1"
noshade="noshade"><h3>Apache Tomcat/5.5.23</h3></body></html>
>>>
```

从上述结果可以看出，服务器返回的状态码是 404，表示找不到请求的 HTML 文档。

下面来看第三种方式的使用，并且也可以通过 putrequest 和 putheader 方法来实现自定义请求，其代码如下：

```
import httplib
print '---------------connetion2-------------'
connection2=httplib.HTTPConnection('www.python.org',80)
connection2.putrequest('GET','/index.html')
connection2.putheader('User-Agent','Mozilla/4.0(compatible;MSIE6.0;Windows
NT5.1)')
connection2.putheader('Accept','*/*')
connection2.endheaders()
re2=connection2.getresponse()
print re2.status,re2.reason
print re2.read(256)
```

在上述代码中，使用 putrequest 的方式来访问特定的 URL 资源，并且使用 putheader 方法设置一些文件头信息，最后通过 getresponse 方法获取相应的 URL 资源，再来看一下执行结果。

```
>>>
--------connection------------
200 OK
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

```
<head>
  <meta http-equiv="content-type" content="text/html; ch
>>>
```

12.4.3 实例描述

如果有一个 URL 资源，但是并不知晓该 URL 是否可用。遇到这样的情况，我一般的做法是：打开 IE 浏览器，测试该地址。由于我喜欢思考，因此并不认为只有这一种方法可以鉴定 URL 资源，苦思冥想之际，本节如何获取 HTML 文档的资源，给我了新的灵感。那就是可以通过某些方法或者属性来获取该 URL 资源的状态码，以此来判断该 URL 是否有效。同样可以将 URL 资源保存到本地，下面就来看一下我的实现方案。

12.4.4 实例应用

【例 12-1】获取 HTML 文档的资源。

(1) 有一个 URL 字符串为 http://localhost/friend.html。

(2) 创建一个名称为 hqHtm.py 的文件，并向其中添加代码。

```
import httplib
import urllib
rsw=raw_input('请输入你想要的操作：直接保存资源(Z)，获得资源的状态码(H)')
if rsw=='H':
    print '--------我是可以获得 URL 资源的状态码方式------------'
    connection=httplib.HTTPConnection('localhost')
    connection.request('GET','/friend.html')
    re=connection.getresponse()
    print re.status,re.reason
    print re.read()
    connection.close()
elif rsw=='Z':
    print '-------我可以执行下载的操作，你可以打开 F://dcy/文件查看哦------'
    url = 'http://localhost/friend.html'
    path='F://dcy/friend.html'
    data = urllib.urlretrieve(url,path)
else:
    print '系统没有为您设置资源'
```

(3) 保存修改好的代码。

12.4.5 运行结果

运行程序，当输入的操作为 H 时，执行结果如下：

```
>>>
请输入你想要的操作：直接保存资源(Z)，获得资源的状态码(H)H
--------我是可以获得 URL 资源的状态码方式------------
200 OK
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>朋友 CGI</title>
#省略部分代码
</body>
</html>
>>>
```

当输入的操作为 Z 时，执行结果如下：

```
>>>
请输入你想要的操作：直接保存资源(Z)，获得资源的状态码(H)Z
-------我可以执行下载的操作，你可以打开 F://dcy/文件查看哦------
>>>
```

打开路径 F:\dcy，结果如图 12-16 所示。

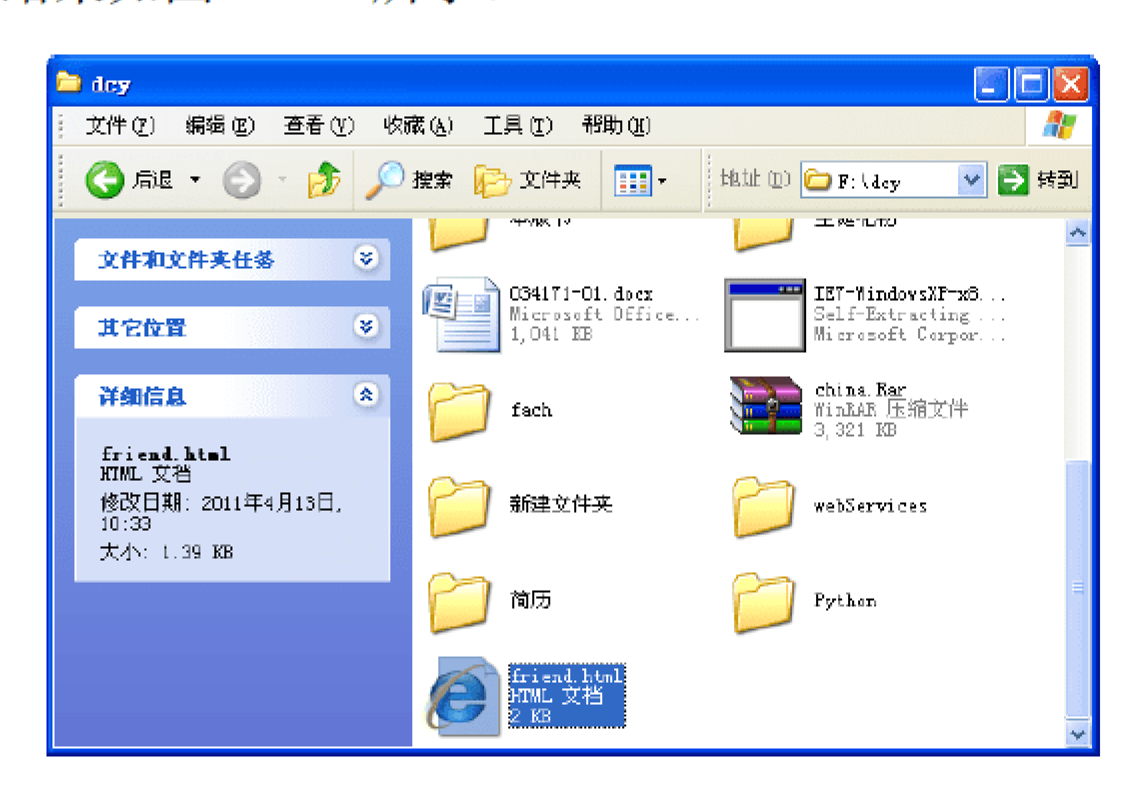


图 12-16　保存URL资源到本地

12.4.6 实例分析

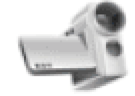


源码解析

在本实例中，导入了 httplib 和 urllib 两大模块，接着使用 httplib 模块的 HTTPConnection 方法返回一个连接对象，然后使用 request 请求 HTML 文档资源，之后调用属性 status 和 reason 来分别获得服务器返回的状态码以及服务器返回该状态码的原因。另外还调用 urllib 模块的 urlretrieve 方法将该文档保存到本机。瞧，这不也是一种判断 URL 资源是否有效的思路吗！

12.5 HTML文档的解析

在获取 URL 资源之后，一般需要对 HTML 文档做进一步处理。在 Python 中，提供了 HTMLParser 模块来解析 HTML 文档。由于 HTML 语言属于 SGML 语言，因此也可以使用 sgmllib 对 HTML 文档进行处理。另外 httpllib 也可以对 HTML 文档进行处理，因为 httpllib 是建立在 sgmllib 模块上的 HTTP 文档高级处理模块。接下来将对这些模块进行详细介绍，首先来看一下 HTMLParser 模块。



视频教学：光盘/videos/12/HTML 文档的解析.avi　长度：10 分钟

12.5.1 基础知识——使用HTMLParser模块

HTMLParser 模块用来解析 HTML 文档，具有小巧、快速和使用简便的特点。该模块采用了一种事件驱动的方式，当模块找到一个特定对象时，可以调用用户自定义的函数进行处理。这里所指的函数是以 handle_开头的 HTMLParser 成员函数。也就是说使用这些函数可以分析 HTML 文档中的标签和数据。接下来我们看一下在 HTMLParser 中定义的 handle_函数，如表 12-9 所示。

表 12-9 HTMLParser中的handle_函数

函数名称	函数用途
handle_starttag	处理开始标签和结束标签在一起的标签，如
handle_startendtag	处理开始的标签
handle_endtag	处理结束的标签
handle_data	处理数据
handle_charref	处理字符实体
handle_entityref	处理实体参考
handle_commant	处理注释

续表

函数名称	函数用途
handle_decl	处理定义
handle_pi	处理“处理指令”

下面通过一个小例子针对表 12-9 中的 handle_data 函数作一详细讲解，其他的均不作介绍，代码如下：

```
import HTMLParser
import urllib
urlText = []
#定义 HTML 解析器
class parseText(HTMLParser.HTMLParser):
    def handle_data(self, data):
        if data != ' ':
            urlText.append(data)
#创建 HTML 解析器的实例
lParser = parseText()
#把 HTML 文件传给解析器
lParser.feed(urllib.urlopen( "http://dict.baidu.com/s?wd=go+go+go&tn=dict")
.read())
lParser.close()
for item in urlText:
    print item
```

在上述代码中，首先将 HTMLParser 模块导入，接下来定义了一个 parseText 类并使其继承 HTMLParser。在类的实现中，重载了 handle_data 函数，当获取传入的 data 数据时，函数 handle_data 会首先判断该数据中是否含有 404 Not Found 或者 Error 404 等字符串，如果含有，则说明该 URL 资源不存在；如果没有，则调用列表对象 urlText 的 append 方法来将数据添加到该列表中。接着声明一个 parseText 实例对象，并调用该实例对象的 feed 方法，从而对获取的 URL 进行处理，最后使用 for 循环将列表 urlText 中的数据输出。最终执行结果如下：

```
>>>
百度词典搜索_go go go
#此处省略部分数据
>>>
```

12.5.2　基础知识——sgmllib的HTML文档处理

sgmllib 模块是 htmllib 模块的基类，主要用来处理 SGML 文档，但同样可以对 HTML 文档进行处理。在 Python 安装路径的 lib 目录下有一个 sgmllib.py 文件，该文件包含了对 sgmllib 模块的实现，并且该文件本身已经提供了简单的 HTML 文档处理功能。当将参数传入到该文件时，将会对参数进行解析，如果没有参数，则会默认处理 test.html 文档。下面来看一下 sgmllib.py 文件如何对 first.html 文档进行处理。

使用 cmd 命令进入 Python 安装路径的 lib 目录下，然后将参数 first.html 文档传入 sgmllib.py 文件，执行结果如图 12-17 所示。

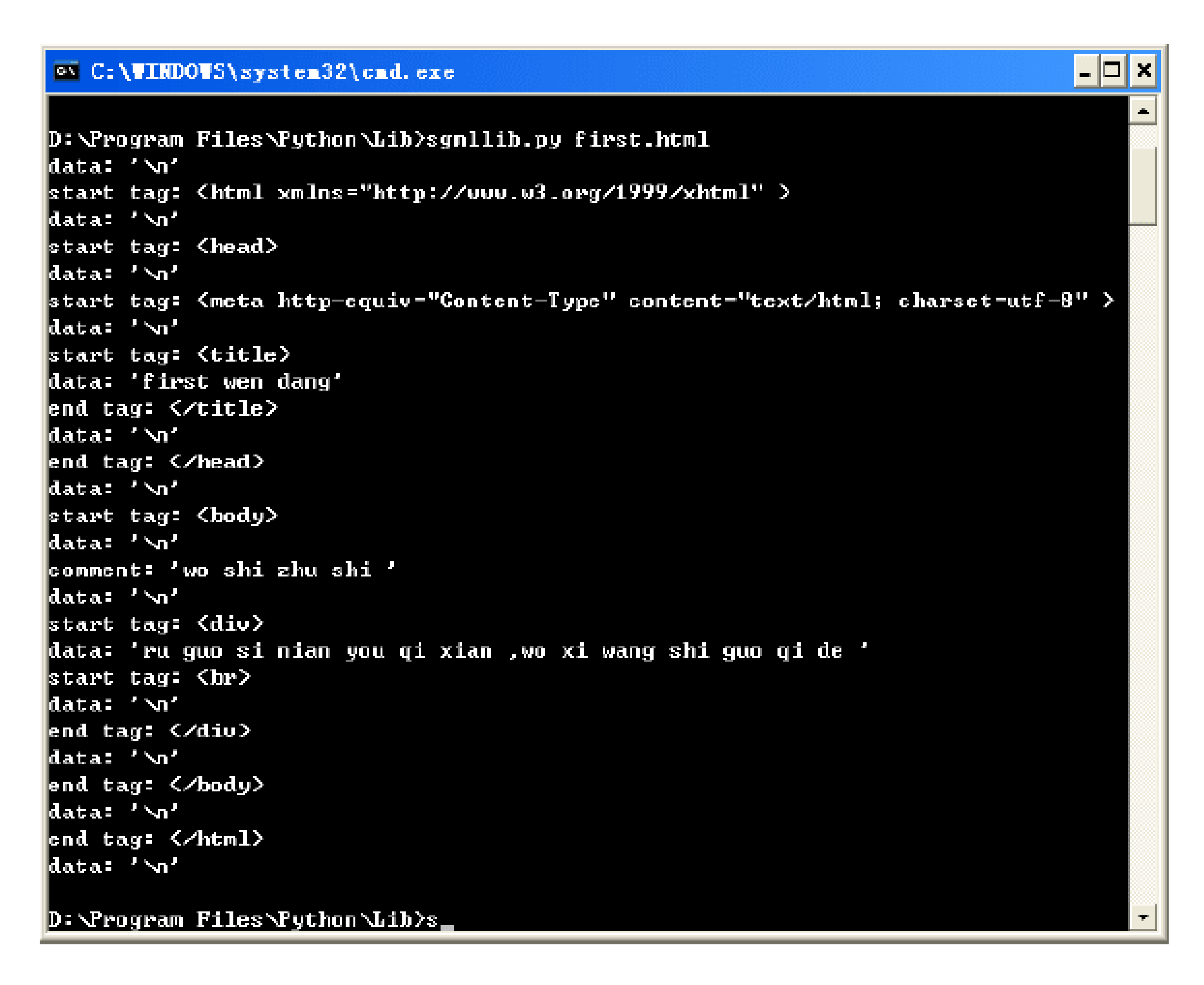


图 12-17　sgmllib处理first.html文档的结果

从图 12-17 可以看出，sgmllib.py 文件对所有标签都进行了识别和输出，主要包括开始标签、结束标签、注释和数据等。另外，我们还看到\n 的数据，这些数据是作为 HTML 文档的一部分被处理的，而在 HTML 文档中并不显示。

之所以会出现图 12-17 所示的结果，是因为在 sgmllib.py 文件中有一个测试类 TestSGMLParser，通过该测试类可以输出部分标签信息。下面就是类 TestSGMLParser 中的代码。

```
class TestSGMLParser(SGMLParser):
    def __init__(self, verbose=0):
        self.testdata = ""
        SGMLParser.__init__(self, verbose)
    def handle_data(self, data):
        self.testdata = self.testdata + data
        if len(repr(self.testdata)) >= 70:
            self.flush()
    def flush(self):
        data = self.testdata
        if data:
            self.testdata = ""
            print 'data:', repr(data)
    def handle_comment(self, data):
        self.flush()
        r = repr(data)
        if len(r) > 68:
            r = r[:32] + '...' + r[-32:]
        print 'comment:', r
    def unknown_starttag(self, tag, attrs):
        self.flush()
        if not attrs:
            print 'start tag: <' + tag + '>'
        else:
            print 'start tag: <' + tag,
            for name, value in attrs:
                print name + '=' + '"' + value + '"',
            print '>'
    def unknown_endtag(self, tag):
        self.flush()
```

```
        print 'end tag: </' + tag + '>'
    def unknown_entityref(self, ref):
        self.flush()
        print '*** unknown entity ref: &' + ref + ';'
    def unknown_charref(self, ref):
        self.flush()
        print '*** unknown char ref: &#' + ref + ';'
    def unknown_decl(self, data):
        self.flush()
        print '*** unknown decl: [' + data + ']'
    def close(self):
        SGMLParser.close(self)
        self.flush()
```

在上述代码中，TestSGMLParser 类继承了 SGMLParser，并且在 TestSGMLParser 类中重载了很多方法，例如 handle_data、handle_comment 和 unknown_starttag 等，从而达到了处理 HTML 文档的目的。

除了可以重载 SGMLParser 类中已有的方法外，还可以使用 start_tag、do_tag 和 end_tag 方式来处理特定的 tag。下面来看一下使用 start_tag 方式处理图片资源的例子，代码如下：

```
import sgmllib
import urllib
class MyLink(sgmllib.SGMLParser):
    def __init__(self):                           #声明一个构造函数
        sgmllib.SGMLParser.__init__(self)
        self.links=[]                             #初始化时将 links 的值设置为空
    def start_img(self,attr):                     #处理 img 标签
        for link in attr:
            tag,attr=link[:2]
            if tag=='src':                        #将取得的 img 资源放入 links 中
                self.links.append(attr)
mylink=MyLink()                                   #实例化一个 MyLink 对象
mylink.feed((urllib.urlopen('http://www.baidu.com')).read())
for position,value in enumerate(mylink.links):        #返回索引的位置和对应的值
    print position,'==>',value
```

在上述代码中，首先将 sgmllib 和 urllib 模块导入，接着创建一个 MyLink 类并使其继承 sgmllib 模块的 SGMLParser。在 MyLink 类中，定义了一个 start_img 方法，在该方法中将<img>标签中 src 的值提取出来，并将其存放到变量 links 中。之后创建一个 MyLink 类的实例 mylink，接着通过 mylink 的 feed 方法对得到的 URL 资源进行处理，最后使用函数 enumerate 分别返回索引位置及其对应的值。下面看一下执行结果。

```
>>>
0 ==> http://www.baidu.com/img/baidu_sylogo1.gif
1 ==> http://gimg.baidu.com/img/gs.gif
>>>
```

12.5.3　基础知识——使用htmllib处理HTML文档

htmllib 模块提供了一种标签驱动 HTML 文档的解析方式，并将解析后的数据传送给 formatter 对象进行处理。由于 htmllib 模块包含 HTMLParser 类，因此通过继承可以有效地对

HTML 文档进行处理。接下来我们通过一个例子如何看一下 htmllib 模块如何获取 HTML 文档中的链接以及更加详细的信息，代码如下：

```
import formatter
import htmllib
import urllib
class MyLink(htmllib.HTMLParser):                    #继承 HTMLParser 类
    def __init__(self):                              #初始化方法
        self.links={}                                #将变量 links 声明成字典类型的结构
        fmatter=formatter.NullFormatter()            #对传来的数据不做任何处理
        htmllib.HTMLParser.__init__(self,fmatter)    #调用初始化方法
    def anchor_bgn(self,href,name,type):             #遇到超链接开始标签时处理
        #将数据放入缓冲区，并不传递给 formatter 解析
        self.save_bgn()
        self.link=href                               #将参数 href 赋给变量 link
    def anchor_end(self):                            #遇到超链接结束标签时处理
        text=self.save_end()
        if self.link and text:
            self.links[text]=self.links.get(text,[])+[self.link]
mylink=MyLink()                                      #实例化一个 MyLink 对象
mylink.feed((urllib.urlopen('http://dict.baidu.com/s?wd=go+go+go&tn=dict'))
.read())                                             #对得到的 URL 资源进行处理
for position,value in mylink.links.items():         #使用 for 循环打印键和值
    print position,'==>',value
```

在上述代码中，首先将 formatter、htmllib 和 urllib 模块导入，接着声明一个类 MyLink 并使其继承 HTMLParser 类。在该类中声明了 3 个方法，分别是__init__、anchor_bgn 和 anchor_end 方法，其中__init__方法为初始化方法，anchor_bgn 和 anchor_end 方法分别对超链接的开始标签、结束标签进行处理。在 anchor_bgn 方法中，调用了本身对象的 save_bgn 方法，其作用是阻止部分数据被传递给 formatter 解析，同样在 anchor_end 方法中，也调用了本身对象的 save_end 方法，目的有两个：一是恢复将数据传递给 formatter 的功能，二是返回缓存中的数据。之后实例化一个 MyLink 对象 mylink，接着调用该对象的 feed 方法对得到的 URL 资源进行处理，最后使用 for 循环将存入 links 字典中的键和值打印输出。执行结果如下：

```
>>>
知道
==> ['http://zhidao.baidu.com/q?ct=17&pn=0&tn=ikaslist&rn=10&word
=go%20go%20go&fr=wwwt']
新闻 ==> ['http://news.baidu.com/ns?cl=2&rn=20&tn=news&word=go%20go%20go']
设置 ==> ['#']
图片
==>
['http://image.baidu.com/i?tn=baiduimage&ct=201326592&lm=-1&cl=2&word=go%20
go%20go']
把百度设为首页 ==> ['#']
视频 ==>
['http://video.baidu.com/v?ct=301989888&rn=20&pn=0&db=0&s=25&word=go%20go%20go']
MP3 ==>
['http://mp3.baidu.com/m?tn=baidump3&ct=134217728&lm=-1&word=go%20go%20go']
到百度词典首页 ==> ['http://dict.baidu.com/']
http://www.wzego.com/... ==> ['http://www.wzego.com/?N,36314.html']
贴吧 ==> ['http://tieba.baidu.com/f?kw=go%20go%20go']
网页 ==> ['http://www.baidu.com/s?wd=go%20go%20go&cl=3&tn=baidu']
```

```
语法标注解释 ==> ['http://www.baidu.com/search/dict.html#n5']
帮助 ==> ['http://www.baidu.com/search/dict.html']
>>>
```

12.6　展示个人小资料

想必读者曾给别人写过纪念、留过言吧，那时候的你是不是对留言簿上各式各样的小资料感到稀奇呢？你有没有提起笔就按照小资料上的指引一步一步填写所需要的信息呢？答案是肯定的。那么在 Python 中也可以展示小资料上的内容，而且还可以上传图片，你心动了吗？前提是：想要实现这样的功能，首先需要了解在 Python 中如何提交表单并且实现文件上传。下面就来看一下在 Python 中如何生成表单页。

**视频教学：**光盘/videos/12/展示个人小资料.avi　　　　长度：4 分钟

12.6.1　基础知识——建立表单页并生成结果页

想必学过 HTML 的人对建立表单页并不陌生，接下来我们使用 HTML 创建一个表单页，使表单提交的路径为/cgi-bin/friend.cgi，其主要代码如下：

```
<body>
<div>
  <h3>从朋友的多少可以测出你以后的命运:<I>你侧过吗？</I></h3>
  <form action="/cgi-bin/friend.cgi">
    <B>输入您的名字:</B>
    <input name="person" value="" size="15"/>
    <p><B>你有多少个朋友？</B></p>
    <input  type="radio" name="howmany" value="0" checked="checked"/>
    我很宅，没有朋友<br />
    <br />
    <input  type="radio" name="howmany" value="10" />
    我不擅长交际，我只有 10 个朋友，但都是知心的哦<br />
    <br />
    <input  type="radio" name="howmany" value="23" />
    自认为不错吧，有 23 个朋友 <br />
    <br />
    <input  type="radio" name="howmany" value="50" />
    我很好哦，我至少有 50 个朋友 <br />
    <br />
    <input  type="radio" name="howmany" value="100" />
    我人缘特好，我至少有 100 个朋友，以诚相待的那种
    <p align="center">
      <input  type="submit" value=" 测 试 "/>
    </p>
  </form>
</div>
</body>
```

输入地址 http://localhost/friend.html，显示效果如图 12-18 所示。

接下来创建一个名称为 friend.cgi 的文件，并添加代码如下：

```
#!D:/Program Files/Python/python.exe
import cgi
reshtml="""Content-Type:text/html\n\n
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>朋友 CGI</title>
</head>
<body>
<div>
<h3>从朋友的多少可以测出你以后的命运:<I>你测过吗？</I></h3>
你输入的名字是：<B>%s</B><br/><br/>
你选择的朋友是：你知道吗？我有<B>%s</B>个朋友
</div>
</body>
</html>
"""
form=cgi.FieldStorage()
who=form['person'].value
howmany=form['howmany'].value
print reshtml%(who,howmany)
```

上述代码是生成结果页的代码，也就是说当按下“测试”按钮之后所转向的脚本。在该脚本中创建了一个 FieldStorage 实例并赋给表单变量 form，该变量中包含表单提交的 person 和 howmany 字段值，接着将这些值分别存入 Python 中的 who 和 howmany 变量中。变量 reshtml 返回 HTML 正文，正文中包含一些动态填好的字段，这些字段的数据都是从表单中读取的。

在地址栏输入 http://localhost/friend.html 浏览页面，然后在名字的文本框中输入内容，其效果如图 12-19 所示。

单击“测试”按钮，显示效果如图 12-20 所示。

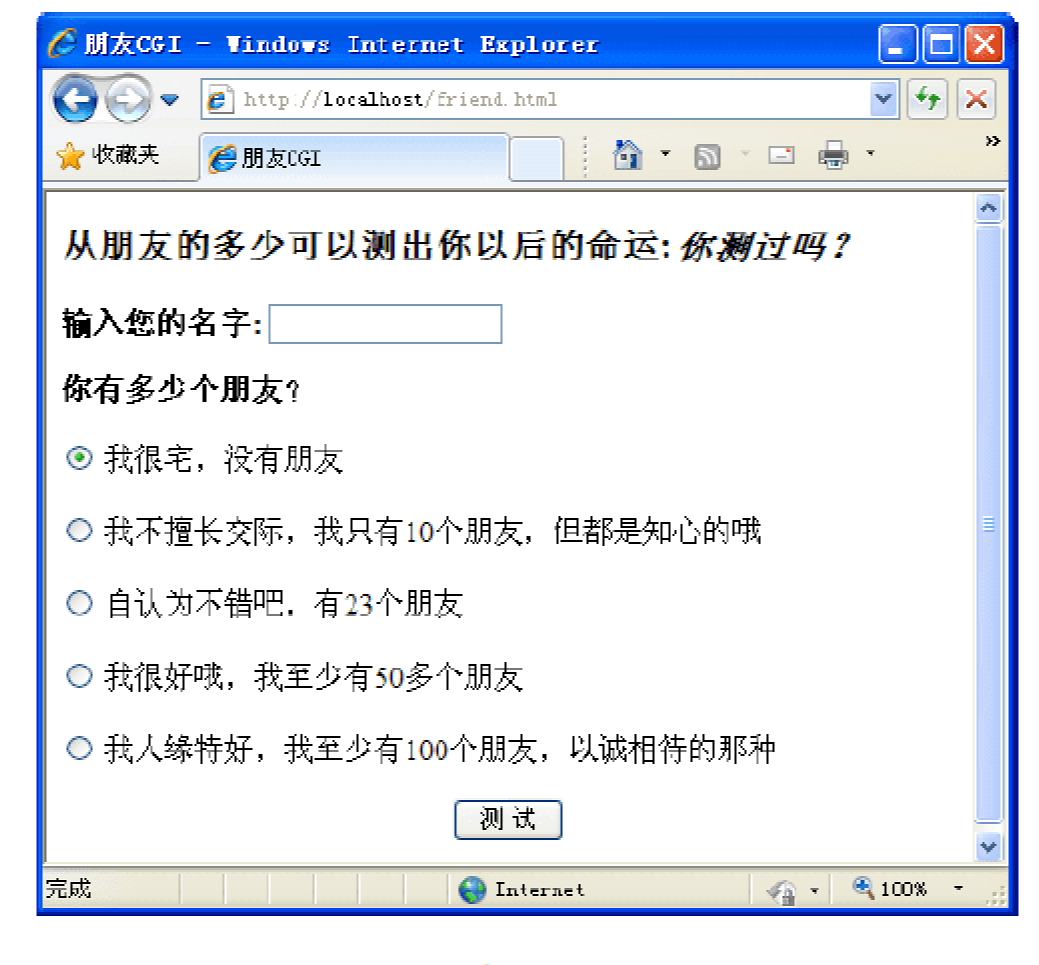


图 12-18　建立的HTML页面

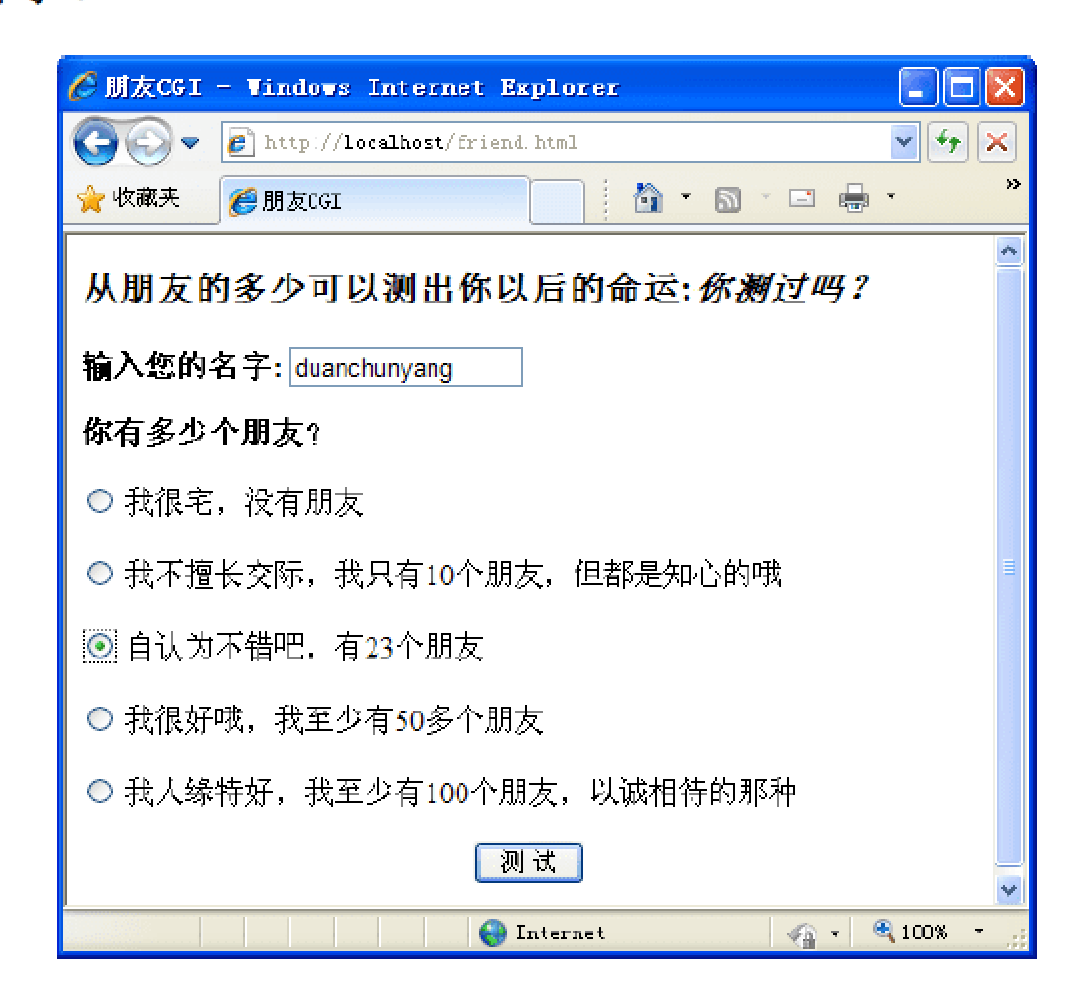


图 12-19　输入内容并选择单选按钮

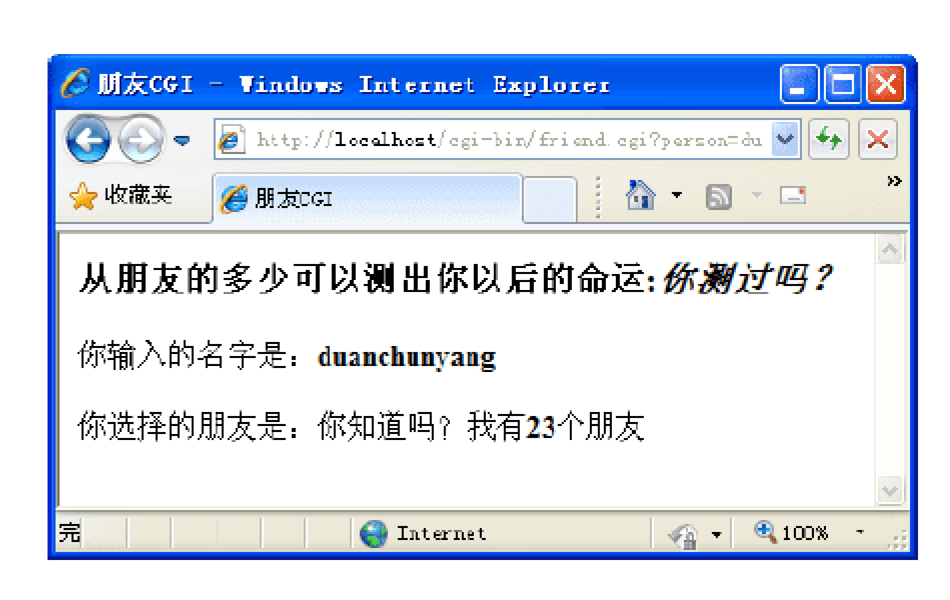


图 12-20　提交之后的结果页显示

12.6.2　基础知识——生成表单和结果页面

在前面使用 HTML 建立的表单页，能否在 CGI 脚本中动态生成该表单页和结果页呢？答案是肯定的。接下来看一下如何实现。

首先需要生成一个用户可以输入数据的表单页面，如果该表单数据有发送的情况，那就需要建立一个结果页面。创建一个名称为 friends.cgi 文件，其代码如下：

```
#!D:/Program Files/Python/python.exe
import cgi
header="Content-Type:text/html\n\n"        #将 HTTP MIME 头从 HTML 正文中提取
#输入数据的表单页面
formhtml="""
<html>
<head>
<title>朋友 CGI</title>
</head>
<body class="bg">
<div class="bground">
  <h3>从朋友的多少可以测出你以后的命运:<I>你测过吗？</I></h3>
  <form action="/cgi-bin/friends.cgi">
    <B>输入您的名字:</B>
    <input name="action" type="hidden" value="edit" />
    <input name="person" value="" size="15"/>
    <p><B>你有多少个朋友？</B></p>
    %s
     <p align="center">
      <input  type="submit" value=" 测 试 "/>
    </p>
  </form>
</div>
</body>
</html>
"""
#声明一个单选按钮的变量
radio="<input  type=radio name=howmany value='%s' %s />%s\n"
#负责用户输入，生成表单页的函数
def showForm():
    friends=""
    for i in [0,10,25,50,100]:
```

```
        checked=""
        if i==0:
            checked="checked"
        friends=friends+radio %(str(i),checked,str(i))
    print header+formhtml %(friends)        #为数据表单页添加 MIME 头信息
#生成结果页面
reshtml="""
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>朋友 CGI</title>
</head>
<body>
<div>
  <h3>从朋友的多少可以测出你以后的命运:<I>你测过吗？</I></h3>
  你输入的名字是：<B>%s</B><br/>
  <br/>
  你选择的朋友是：你知道吗？我有<B>%s</B>个朋友 </div>
</body>
</html>
"""
#负责生成表单页的函数
def doResults(who,howmany):
    print header+reshtml % (who,howmany)        #为结果页面添加 MIME 头信息
#具体执行某一步操作
def process():
    form=cgi.FieldStorage()
    if form.has_key('person'):
       who=form['person'].value
    else:
        who='NULLUSER'
    if form.has_key('howmany'):
        howmany=form['howmany'].value
    else:
        howmany=0
    if form.has_key('action'):
        doResults(who,howmany)
    else:
        showForm()
if __name__=="__main__":
   process()
```

在上述代码中，将 HTTP 的 MIME 类型的头部信息从 HTML 正文中提取出来并赋给 header 变量，接着声明一个用户输入数据的表单页面 formhtml 和结果页面 reshtml，而在 formhtml 中有一个 name 为 action 的 input 元素，将其类型设置为 hidden，值为 edit，表示具体显示表单页面或者结果页面。之后声明一个单选按钮的变量 radio，接着声明一个负责生成表单页的函数 showForm 和负责生成结果页的函数 doResults，最后通过函数 process 说明具体执行的操作。



千万不要忘记修改 D:\Apache2.2\conf 文件中的 httpd.conf 文件，在<Directory>标记中添加代码 PythonHandler friends，否则会出现错误。

输入地址 http://localhost/cgi-bin/friends.cgi 进行浏览，执行结果如图 12-21 所示。



图 12-21　浏览页面时出错

从图 12-20 可以看出，执行结果出现了错误。这时需要从图 12-20 中提示的 error log 日志文件中找出错误。打开 D:\Apache2.2\logs 文件的 error.log 文件，其中的一段代码如下：

```
[Mon Apr 18 17:02:04 2011] [error] [client 127.0.0.1] SyntaxError: Non-ASCII
character '\\xbd' in file D:/Apache2.2/cgi-bin/friends.cgi
on line 3, but no encoding declared; see
http://www.python.org/peps/pep-0263.html for details\r
```

在上述日志代码中，从 Non-ASCII 可以看出是字符编码的错误，因此只需将 D:\Apache2.2\cgi-bin 文件下的 friends.cgi 文件另存为 utf-8 的格式即可。

再次打开浏览器，输入 http://localhost/cgi-bin/friends.cgi，执行结果如图 12-22 所示。

在“输入您的名字”文本框中输入“蓝色忧郁”，并选择 25 单选按钮，然后单击“测试”按钮，执行结果如图 12-23 所示。

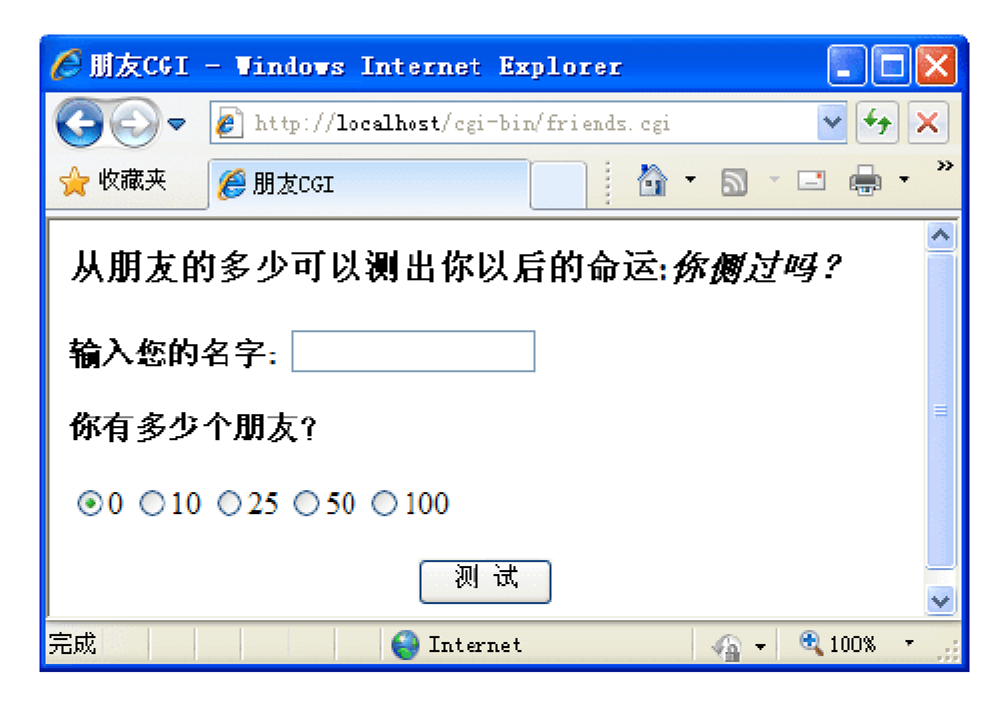


图 12-22　生成表单页

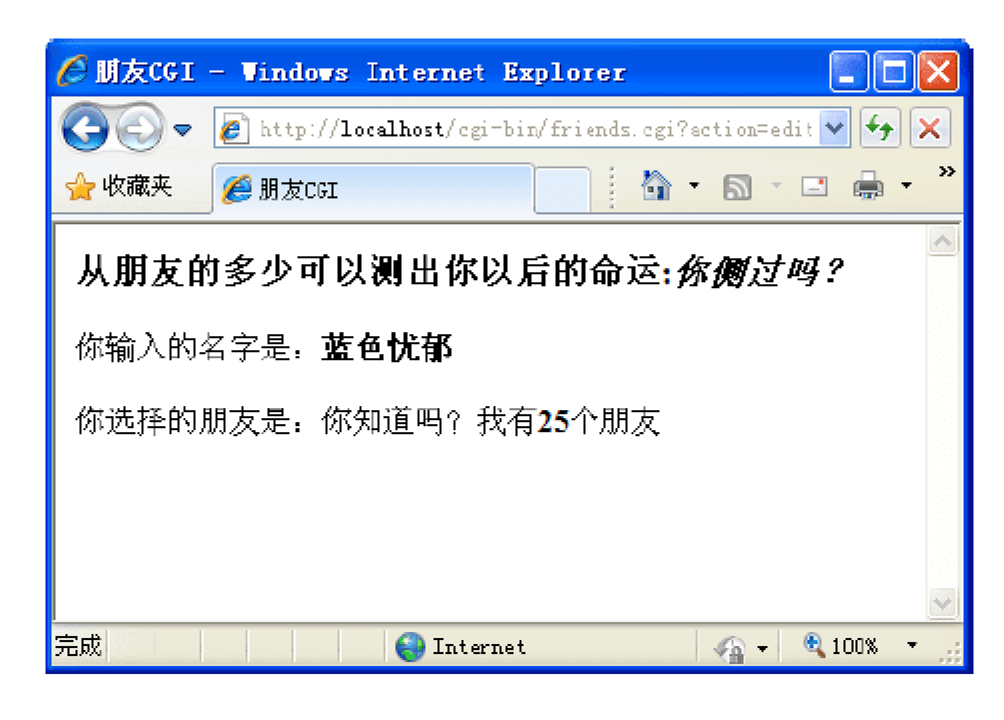


图 12-23　提交到结果页

12.6.3　基础知识——完善表单页和结果页

前面只简单介绍了如何在 CGI 脚本中生成表单页并提交到结果页，如果我们没有选择单选按钮，指明朋友的数量，那么该单选按钮的值就不会被传到服务器，该怎么办？当用户输入的数据被提交到结果页时，该页面如何通知用户做下一步操作？本节将针对这些缺点一一完善以便能给用户一个完整的 Web 应用体验。

接下来看一下完善表单页和结果页代码的步骤。

(1) 重新建立一个名称为 friendWeb.cgi 的文件，并将头部代码从 friends.cgi 脚本中复制到 friendWeb.cgi 中。需要复制的代码如下：

```
#!D:/Program Files/Python/python.exe
import cgi
header="Content-Type:text/html\n\n"        #将 HTTP MIME 头从 HTML 正文中提取
```

(2) 如果用户没有选择单选按钮，则我们可以创建一个错误页面并赋给变量 errorHtml。在该页面中使用 JavaScript 代码使按钮实现返回的功能，之后声明一个 showError 函数将错误页面返回到客户端。其代码如下：

```
#出错的页面
errorHtml="""
<html>
<head>
<title>朋友 CGI</title>
</head>
<body class="bg">
<H3>不好意思！出错了！！</H3>
<B>%s</B>
<form>
  <input type="button" value=" 返 回 " onclick="window.history.back()" />
</form>
</body>
</html>
"""
#负责显示错误页面的函数
def showError(error_str):
    print header+errorHtml %(error_str)
```

(3) 声明一个表单页面，该表单页面与 friends.cgi 文件不同，这里将表单页面提交的地址作为参数传入到 formhtml 中，代码如下：

```
#声明表单页面
url="/cgi-bin/friendWeb.cgi"
formhtml="""
<html>
<head>
<title>朋友 CGI</title>
</head>
<body class="bg">
<div class="bground">
  <h3>从朋友的多少可以测出你以后的命运:<I>你测过吗？</I></h3>
  <form action="%s">
    <B>输入您的名字:</B>
    <input name="action" type="hidden" value="edit" />
    <input name="person" value="" size="15"/>
    <p><B>你有多少个朋友？</B></p>
    %s
     <p align="center">
      <input  type="submit" value=" 测 试 "/>
    </p>
  </form>
</div>
</body>
```

```
</html>
"""
#声明一个单选按钮的变量
radio="<input  type=radio name=howmany value='%s' %s />%s\n"

#负责用户输入，生成表单页的函数
def showForm(who,howmany):
    friends=""
    for i in [0,10,25,50,100]:
        checked=""
        if str(i)==howmany:
            checked="checked"
        friends=friends+radio %(str(i),checked,str(i))
    print header+formhtml %(url,friends)        #为数据表单页添加 MIME 头信息
```

(4) 接下来生成结果页的函数，在结果页中有一个链接，作用是返回输入数据页面，其链接地址由基地址 url 和传入的参数拼接而成。代码如下：

```
#生成的结果页面
reshtml="""
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>朋友 CGI</title>
</head>
<body>
<div>
  <h3>从朋友的多少可以测出你以后的命运:<I>你测过吗？</I></h3>
  你输入的名字是：<B>%s</B><br/>
  <br/>
  你选择的朋友是：你知道吗？我有<B>%s</B>个朋友 </div>
  <p><a href='%s'>单击这里</a>返回到开始页面</p>
</body>
</html>
"""

#负责生成结果页的函数
def doResults(who,howmany):
    newurl=url+'?action=reedit&person=%s&howmany=%s'%(who,howmany)
    print header+reshtml % (who,howmany,newurl)        #为结果页面添加 MIME 头信息
```

(5) 根据不同的选择，执行不同的操作，其代码如下：

```
#具体执行某一步的操作
def process():
    error=''
    form=cgi.FieldStorage()
    if form.has_key('person'):
       who=form['person'].value
    else:
        who='NULLUSER'
    if form.has_key('howmany'):
        howmany=form['howmany'].value
    else:
        if form.has_key('action')and form['action'].value=='edit':
            error='请选择您有几个好朋友'
        else:
```

```
            howmany=0
    if not error:
        if form.has_key('action') and form['action'].value!='reedit':
            doResults(who,howmany)
        else:
            showForm(who,howmany)
    else:
        showError(error)
if __name__=='__main__':
    process()
```

(6) 打开 D:\Apache2.2\conf 文件中的 httpd.conf 文件，在<Directory>标记中添加代码 PythonHandler friendWeb。

(7) 保存修改好的代码。

打开浏览器，在地址栏输入 http://localhost/cgi-bin/friendWeb.cgi，并在名字的文本框上输入内容，如图 12-21 所示。

(8) 单击“测试”按钮，执行效果如图 12-24 所示。

(9) 单击图 12-25 中的“返回”按钮，并选择 25 单选按钮，然后单击“测试”按钮，执行结果如图 12-26 所示。

(10) 单击“单击这里”超链接，显示结果如图 12-27 所示。

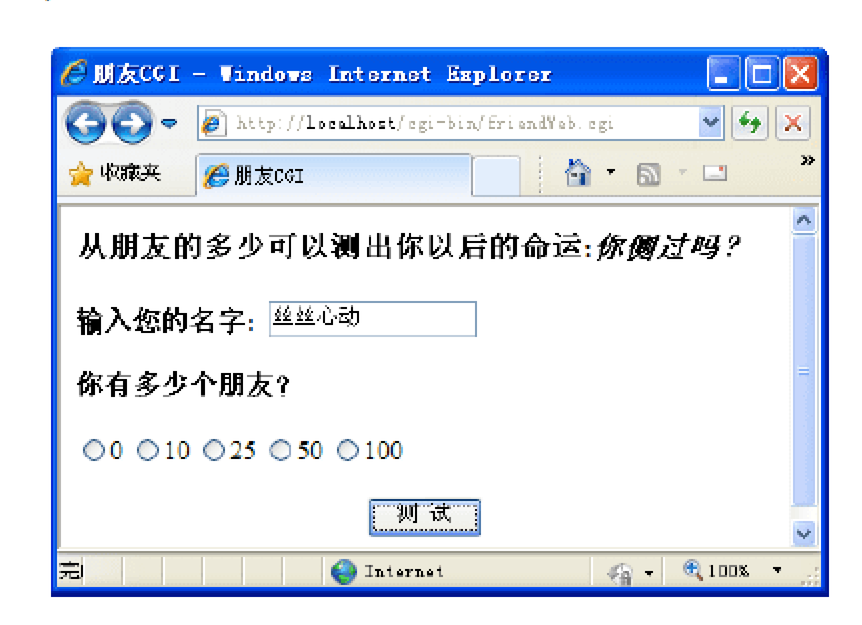


图 12-24 输入“丝丝心动”文本

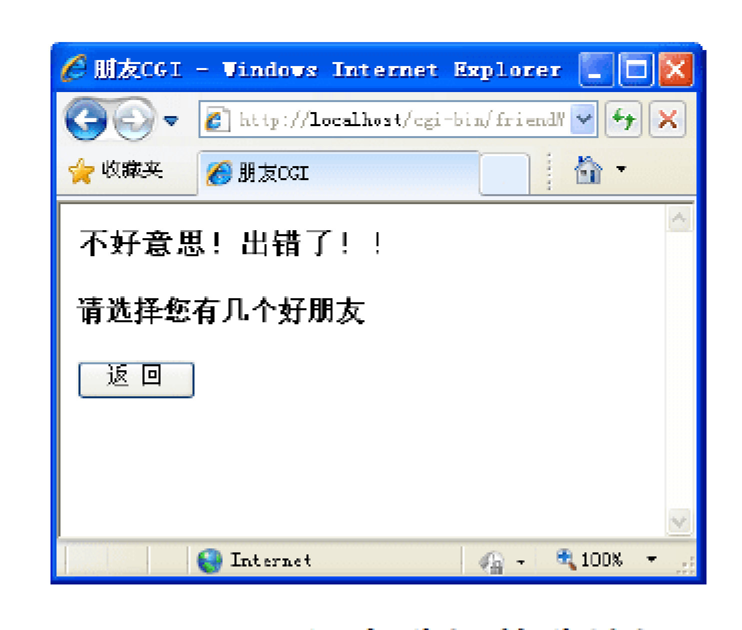


图 12-25 没有选择单选按钮

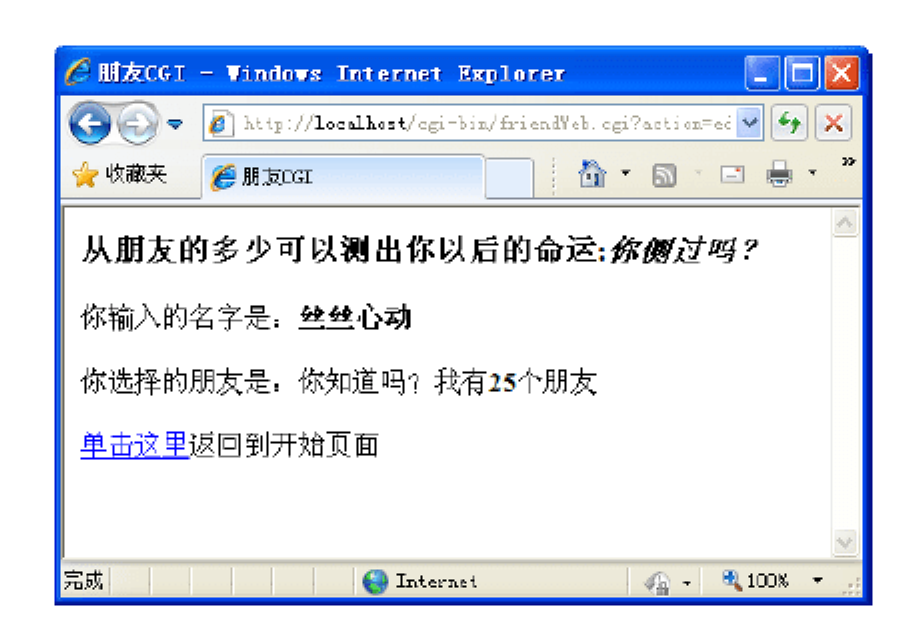


图 12-26 提交成功时

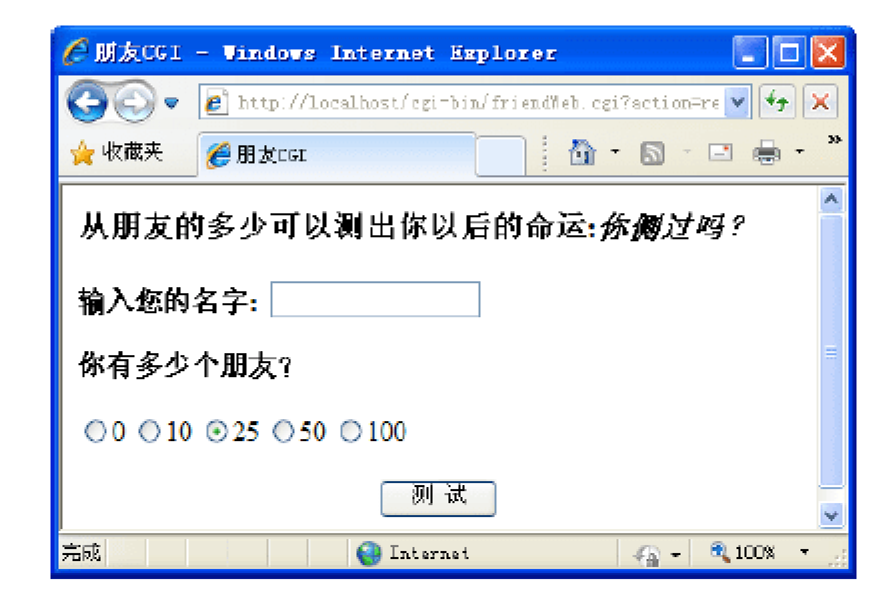


图 12-27 单击超链接返回

到目前为止，一个完整的 Web 应用体验就展示到用户面前了。

12.6.4　基础知识——Mulitipart表单提交和文件上传

到目前为止，CGI 只允许两种表单编码，分别是 application/x-www-form-urlencoded 和 multipart/form-dat。由于 application/x-www-form-urlencoded 编码是默认的，因此在 Form 标签中不必声明，而对于 multipart 表单，则需要明确指出编码方式。

```
<form enctype="multipart/form-data">  </from>
```

提交表单时使用 multipart 编码方式可以实现文件上传。下面我们通过一个例子来看一下 multipart 的使用，代码如下：

```
#!D:/Program Files/Python/python.exe
import cgi, os, urllib, md5
print "Content-type: text/html"
print
print """<HTML>
<HEAD>
<TITLE>文件上传</TITLE></HEAD><BODY>"""
form = cgi.FieldStorage()
if form.has_key('file'):
    fileitem = form['file']
    if not fileitem.file:
        print "出错啦！您上传的文件出错了<P>"
    else:
        print "您上传的文件是：%s<P>" % cgi.escape(fileitem.filename)
        m = md5.new()
        size = 0
        while 1:
            data = fileitem.file.read(4096)
            if not len(data):
                break
            size += len(data)
            m.update(data)
            filename=os.path.split(fileitem.filename) #获得上传文件的名称

            file(filename[1],'wb').write(data) #将上传的文件写入到 data 文件中
        print "接收到的文件有 %d 字节，上传的路径是：%s" %(size,os.environ['SCRIPT_NAME'])
else:
    print "您还没有上传任何文件.<P>"
print """<FORM METHOD="POST" ACTION="%s" enctype="multipart/form-data">
请选择要上传的文件：<INPUT TYPE="file" NAME="file"><br/>
""" % os.environ['SCRIPT_NAME']
print """<INPUT TYPE="submit" NAME="submit" VALUE=" 上 传 ">
</FORM>
</BODY></HTML>"""
```

在上述代码中，首先将 cgi、os、urllib、md5 这些模块导入，接着使用 Python 描述 HTML 文档显示的内容，之后创建 FieldStorage 的实例以此来获取 form 表单中上传的文件，并将其内容写入 data 文件，然后使用 len 方法获取上传文件的字节数，并使用 os.environ['SCRIPT_NAME'] 获取文件上传的路径。

在浏览器的地址栏输入 http://localhost/cgi-bin/Multipart.cgi，执行结果如图 12-28 所示。

选中要上传文件，单击“上传”按钮，执行结果如图 12-29 所示。

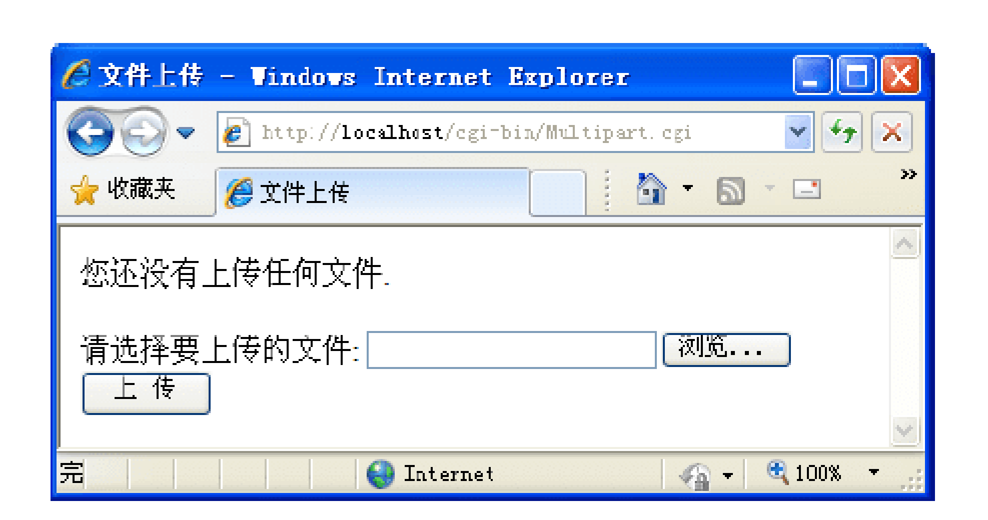


图 12-28 浏览页面

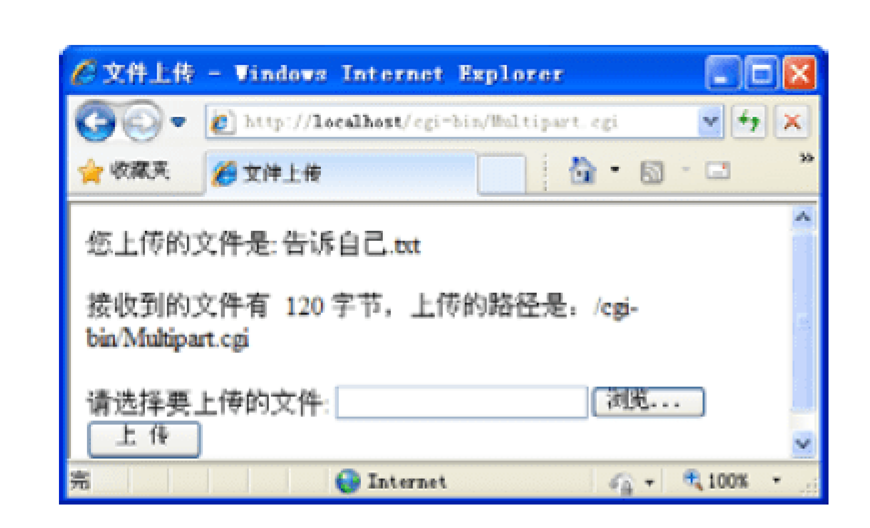


图 12-29 文件上传成功

12.6.5 实例描述

《同桌的你》这首歌使多少人想起高中时代很要好的同学。与之不同的是我偶然看留言簿，想起了那一群为同一个目标共同努力，为相聚在那里一起开心，因毕业分离而抱在一起哭泣的同学，那一张张笑脸在我脑海中闪过，顿时热泪盈眶。为纪念那些令我感动的人，我制作了一个小资料，将他们的信息分别录入。等到年老的时候，我想这肯定是一份不可多得的宝藏。下面来看一下我的实现方案。

12.6.6　实例应用

【例 12-2】展示个人小资料。

(1) 创建一个 HTML 页面，其主要代码如下：

```
<h3>偶滴小资料</h3>
  <form action="/cgi-bin/ziliao.cgi" enctype="multipart/form-data">
    <p><B>姓名：</B>
      <input name="name"  value=""  size="20"/>
    </p>
    <p><B>生日：</B>
      <input name="birthday" value="" />
    </p>
    <p><B>爱好：</B>
      <input name="aihao" value="" />
    </p>
    <p><B>喜欢的颜色：</B>
      <input name="yanse" value="" />
    </p>
    <p><B>喜欢的节日：</B>
      <input name="jieri" value="" />
    </p>
    <p><B>最伤心的事：</B>
      <input name="shangxin" value="" />
    </p>
    <p><B>心情谁分享：</B>
      <input name="xinqing" value="" />
    </p>
    <p><B>爱生活方式：</B>
      <input name="shenghuo" value="" />
    </p>
    <p align="center">
      <input  type="submit" value=" 提 交 "/>
    </p>
  </form>
```

(2) 创建一个名称为 ziliao.cgi 的文件，并在该文件中添加如下代码：

```
#!D:/Program Files/Python/python.exe
import cgi
reshtml="""Content-Type:text/html\n\n
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<title>个人小资料</title>
</head>
<body background="img/http_imgload12.jpg">
<div>
<h3>偶滴小资料</h3>
你的名字是：<B>%s</B><br/><br/>
你的生日是：<B>%s</B><br/><br/>
你的爱好是：<B>%s</B><br/><br/>
你喜欢的颜色是：<B>%s</B><br/><br/>
```

```
你喜欢的节日是：<B>%s</B><br/><br/>
你最伤心的事是：<B>%s</B><br/><br/>
你的心情与谁分享是：<B>%s</B><br/><br/>
你喜欢的生活方式是：<B>%s</B><br/><br/>
</div>
</body>
</html>
"""
form=cgi.FieldStorage()
name=form['name'].value
birthday=form['birthday'].value
aihao=form['aihao'].value
yanse=form['yanse'].value
jieri=form['jieri'].value
shenghuo=form['shenghuo'].value
shangxin=form['shangxin'].value
xinqing=form['xinqing'].value
print reshtml%(name,birthday,aihao,yanse,jieri,shangxin,xinqing,shenghuo)
```

(3) 打开 D:\Apache2.2\conf 文件夹下的 httpd.conf 文件，找到<Directory>标签并添加代码如下：

```
PythonHandler ziliao
```

(4) 打开 D:\Apache2.2\cgi-bin 文件夹下的 ziliao.cgi 文件，将其另存为 utf-8 格式的文件。

(5) 保存修改的代码。

12.6.7　运行结果

在浏览器地址栏中输入http://localhost/ziliao.html，执行结果如图12-30所示。接着在图12-30所示页面中输入内容并单击“提交”按钮，执行结果如图 12-31 所示。

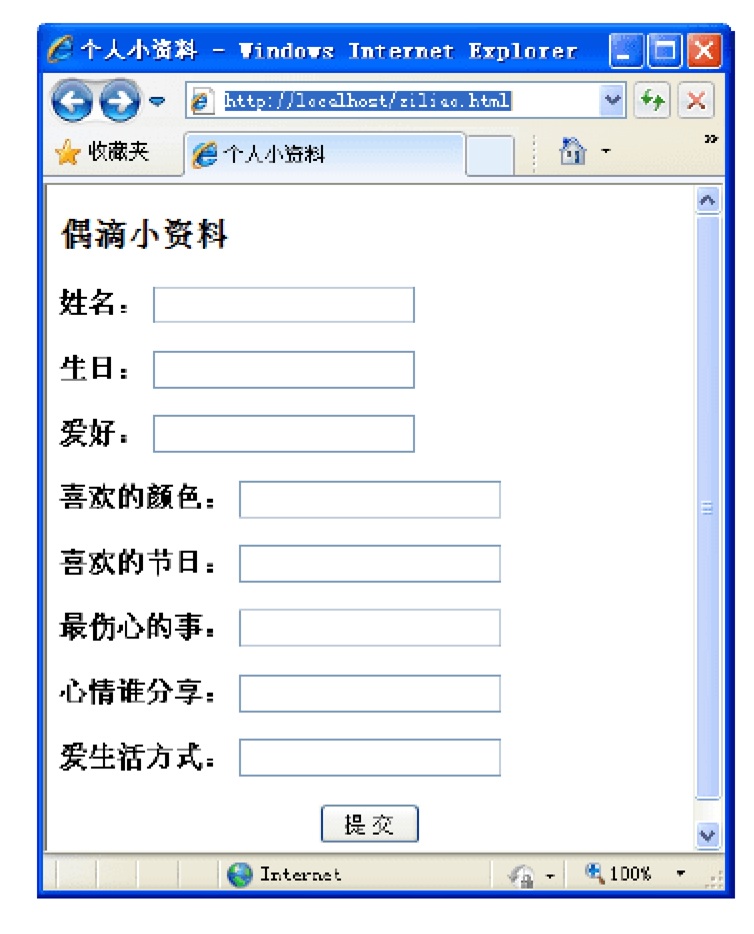


图 12-30　个人资料表单页面

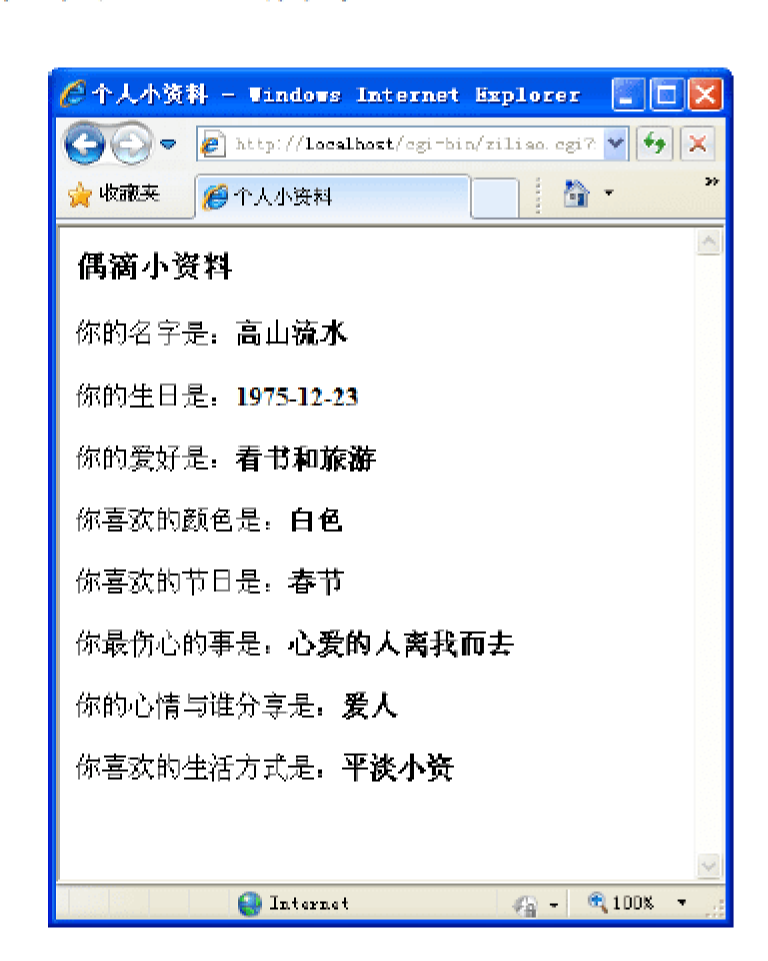


图 12-31　从表单获取内容

12.6.8　实例分析



源码解析

在本实例中，首先创建一个表单页面，接着将该表单中的内容提交到 ziliao.cgi 脚本文件，而在该脚本文件中创建了一个 FieldStorage 实例 form，并使用 form['name'].value 的方式获取用户在表单页面输入的值，最后将这些值传入到已经声明的 reshtml 文档中。瞧，这样一个简单的个人小资料就形成了，是不是很简单呢！

12.7　常见问题解答

12.7.1　Python中的urlopen问题



Python 中的 urlopen 问题。
网络课堂：http://bbs.itzcn.com/thread-15811-1-1.html

我有这样一段代码：

```
#!/usr/bin/python
# Filename: for.py
import urllib
for i in range(1,10):
       doc = urllib.request.urlopen("http://www.baidu.com").read()
else:
       print('loop over!')
```

我的运行环境是 Python 2.5，运行结果总是提示下面的错误代码。

```
Traceback (most recent call last):
  File "F:/Python/第 12 章/9.py", line 5, in <module>
    doc = urllib.request.urlopen("http://www.baidu.com").read()
AttributeError: 'module' object has no attribute 'request'
```

这到底是怎么回事？请高手指点一二。

【解决办法】很高兴为你解答。

在 urllib 模块中有一个方法 urlopen，用于获取 URL 资源，因此直接使用 urllib.urlopen 方法即可。修改代码如下：

```
#!/usr/bin/python
# Filename: for.py
import urllib
for i in range(1,10):
       doc = urllib.urlopen("http://www.baidu.com").read()
else:
       print('loop over!')
```

执行结果如下：

```
>>>
loop over!
>>>
```

瞧，这是你想要的结果吧。

12.7.2 Python中的urllib2 问题



Python 中的 urllib2 问题。
网络课堂：http://bbs.itzcn.com/thread-15812-1-1.html

我使用 urllib2 模块的 urlopen 方法打开一个 URL 资源，执行结果出现错误。下面就是我的代码。

```
import urllib2
handler=urllib2.urlopen('''http://www.google.com/translate_a/t?client=t&sl=
zh-CN&tl=en&q=%E4%BD%A0%E5%A5%BD''')
```

运行程序，执行出现的错误代码如下：

```
Traceback (most recent call last):
  File "F:/Python/第 12 章/9.py", line 2, in <module>

handler=urllib2.urlopen('''http://www.google.com/translate_a/t?client=t&sl=
zh-CN&tl=en&q=%E4%BD%A0%E5%A5%BD''')
  File "D:\Program Files\Python\lib\urllib2.py", line 124, in urlopen
    return _opener.open(url, data)
  File "D:\Program Files\Python\lib\urllib2.py", line 387, in open
    response = meth(req, response)
  File "D:\Program Files\Python\lib\urllib2.py", line 498, in http_response
    'http', request, response, code, msg, hdrs)
  File "D:\Program Files\Python\lib\urllib2.py", line 425, in error
    return self._call_chain(*args)
  File "D:\Program Files\Python\lib\urllib2.py", line 360, in _call_chain
    result = func(*args)
  File "D:\Program Files\Python\lib\urllib2.py", line 506, in
http_error_default
    raise HTTPError(req.get_full_url(), code, msg, hdrs, fp)
HTTPError: HTTP Error 403: Forbidden
```

这是为什么？请高手指点！

【解决办法】很高兴为你解答。

你的程序之所以会出现 403 错误，是因为你访问的 google 地址做了限制，不允许爬虫访问该网页，你只需伪装成浏览器就可以了，其代码如下：

```
from urllib import FancyURLopener
class MyOpener(FancyURLopener):
    version = 'Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.8.1.11)
Gecko/20071127 Firefox/2.0.0.11'
myopener = MyOpener()
page = myopener.open('http://www.google.com/translate_a/t?client=t&sl
=zh-CN&tl=en&q=%E4%BD%A0%E5%A5%BD')
print page.read()
```

运行程序，执行结果如下：

```
>>>
[[["Hello","浣犲ソ","","N 菓 h 羹
o"]],[["interjection",["hello","hi","hallo"]],["phrase",["How are
you"]]],"zh-CN",,[["Hello",[5],1,0,919,0,1,0]],[["
浣犲ソ",4,,,""],["浣犲ソ",5,[["Hello",919,1,0],["Hi",49,1,0]],[[0,2]],"浣犲ソ
"]],,,[],3]
>>>
```

这样就能访问该地址了。

12.8 习　　题

一、填空题

(1) urlparse 模块主要负责 URL 字符串的解析、拼接等功能，那么我们使用__________方法可以将 URL 字符串拆分成一个 6 元组。

(2) 有这样一段代码：

```
import urlparse
print "\n 通过拼接子路径来生成 Python 文档页面的 URL"
newURL =
__________.urljoin('http://www.bai.com/admin/',"module-urllib2/request-
objects.html")
print 'newURL 的地址是：',newURL
```

在上述代码的空白处填写__________方能使运行结果如下：

```
通过拼接子路径来生成 Python 文档页面的 URL
newURL 的地址是：
http://www.bai.com/admin/module-urllib2/request-objects.html
```

(3) 在 Python 的 urllib 模块中有两种获取网络资源的方式，分别是：__________和 urlretrieve。

(4) 在 Python 中，我们除了可以使用 urllib 和 urllib2 模块获取 HTML 文档的资源外，还可以通过__________模块来获取文档资源。

(5) 众所周知，使用 HTMLParser 模块可以解析 HTML 文档，那么使用该模块中的__________函数来处理 HTML 文档中的数据。

二、选择题

(1) 在对 URL 进行处理时，我们使用 join 方法将 URL 拼接成新的字符串，使用该方法的代码如下：

```
import urlparse
print "\n 通过拼接子路径来生成 Python 文档页面的 URL"
newURL =
urlparse.urljoin('http://www.bai.com/dcy/',"request-objects.html")
XURL = urlparse.urljoin('http://www.bai.com/dcy',"request-objects.html")
print 'newURL 的地址是：',newURL
```

```
print 'XURL 的地址是：',XURL
```

运行程序，执行结果是下面选项中的__________。

A.

```
>>>
通过拼接子路径来生成 Python 文档页面的 URL
newURL 的地址是： http://www.bai.com/dcy/request-objects.html
XURL 的地址是： http://www.bai.com/request-objects.html
>>>
```

B.

```
>>>
通过拼接子路径来生成 Python 文档页面的 URL
newURL 的地址是： http://www.bai.com/dcy/request-objects.html
XURL 的地址是： http://www.bai.com/dcyrequest-objects.html
>>>
```

C.

```
>>>
通过拼接子路径来生成 Python 文档页面的 URL
newURL 的地址是： http://www.bai.com/dcy/request-objects.html
XURL 的地址是： http://www.bai.com/dcy/request-objects.html
>>>
```

D.

```
>>>
通过拼接子路径来生成 Python 文档页面的 URL
newURL 的地址是： http://www.bai.com/request-objects.html
XURL 的地址是： http://www.bai.com/request-objects.html
>>>
```

(2) urlparse 模块主要负责对 URL 字符串进行解析、拼接等功能，在 urlparse 模块中提供了几种方法，分别可以实现：将 URL 字符串解析成一个 6 元组，将一个 6 元组合成一个字符串，以及将该 URL 字符串拼接成一个新字符串，它们分别是__________。

A. urlparse()、urljoin()和 urlunparse()　　B. urlparse()、urlunparse()和 urljoin()

C. urlunparse()、urlparse()和 urljoin()　　D. urljoin()、urlunparse()和 urlparse()

(3) 在使用 HTMLParser 模块解析 HTML 文档时，需要使用下面选项中的__________函数来处理 HTML 文档的数据。

A. handle_starttag　　B. handle_commant

C. handle_data　　D. handle_endtag

(4) 在 Python 中提交表单时，我们可以使用某种编码方式实现文件上传。在下面几个选项中，其编码方式正确的是__________。

A. <form enctype="application/x-www-form-urlencoded">　</from>

B. <form enctype="x-www-form-urlencoded">　</from>

C. <form enctype="form-data">　</from>

D. <form enctype="multipart/form-data">　</from>

三、上机题

上机练习： 实现登录窗口。

通过对本章的学习，我们了解了使用 urlparse 模块可以实现对 URL 字符串解析、拼接等功能，获取 HTML 文档资源的几种方式以及如何解析 HTML 文档。另外，本章还介绍了帮助 Web 服务器处理客户端数据的 CGI 技术，这将在以后的程序开发中起到很重要的作用，因此本章的上机练习就是针对 CGI 技术。

有一个登录窗口，其运行效果如图 12-32 所示。当你输入的用户名和密码不是 helloworld，并单击“提交”按钮时，执行效果如图 12-33 所示。



图 12-32　登录窗口

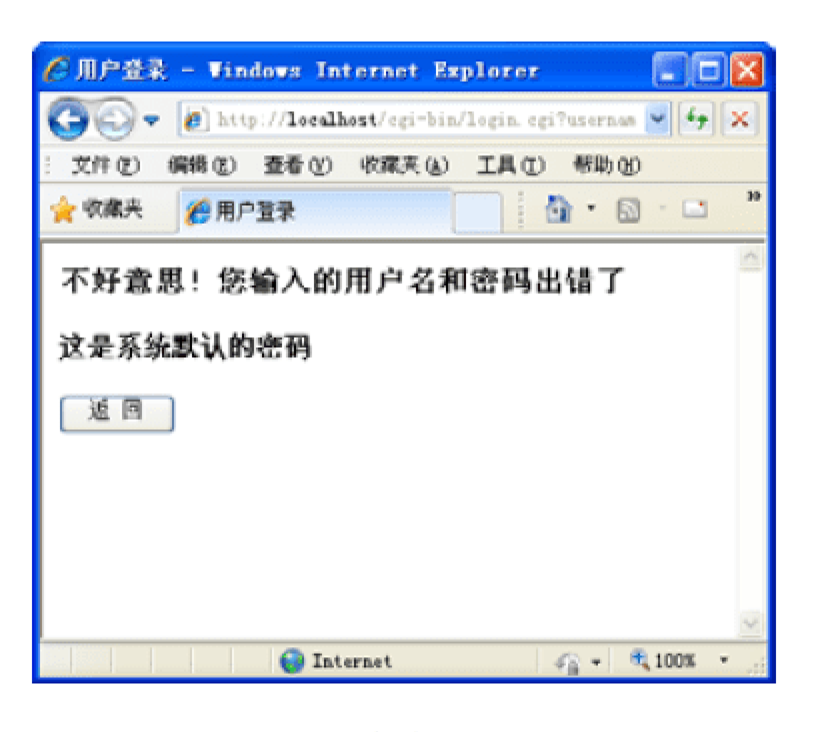


图 12-33　用户名或者密码错误

当输入的用户名和密码是 helloworld 时，显示效果如图 12-34 所示。

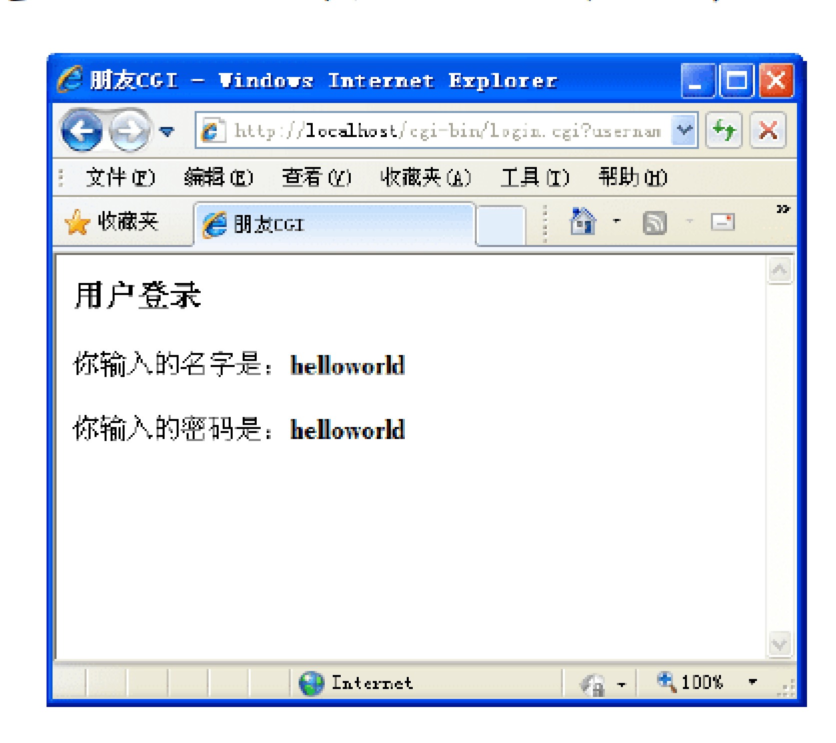


图 12-34　用户名和密码正确



第 13 章　应知应会技能之XML处理

内容摘要

XML 技术已经日益成为当前许多新生技术的核心，在许多领域都有着不同的应用。在 Python 语言中已经包含了对 XML 的支持，本章将介绍如何使用 Python 语言解析 XML 数据。例如：使用 SAX 技术处理 XML 文档，使用 DOM 解析 XML 数据，以及动态生成 XML 内容等。同时本章还将对可扩展样式表语言 XSL 进行详细讲解。

学习目标

- 了解 XML 的概念。
- 掌握 XML 的文档结构。
- 熟练掌握使用 SAX 处理 XML 文档。
- 熟练掌握使用 DOM 解析 XML 数据。
- 了解可扩展样式表语言 XSL。
- 掌握动态生成 XML 内容的方法。

13.1 和我一起学习XML

SGML 是一种描述性语言，即一种描述语言的语言，它定义了以电子形式表示文本的方法。HTML 则是 SGML 的一种应用，它具有通用的语义，适合于表示各种系统域的信息。但是，随着 Internet 技术的发展，你有没有感觉到 HTML 语言的可扩展性不强呢？我想，答案是肯定的。眼下，长江后浪推前浪，专家怎能容忍 HTML 语言的局限性继续存在呢？因此出现了灵活性很高的 XML 标记语言。

下面我们来看一下 XML 的发展史。

XML 从 1996 年开始有其雏形，并向 W3C(全球信息网联盟)提案，而在 1998 年 2 月发布为 W3C 的标准(XML 1.0)。XML 的前身是 SGML(Standard Generalized Markup Language，标准通用标记语言)，是自 IBM 从 20 世纪 60 年代开始发展的 GML(Generalized Markup Language)标准化后的名称。

随着 Web 的广泛应用，W3C 逐渐意识到 HTML 的局限性有以下三点。

- 不能解决所有解释数据的问题：像影音文件、化学公式、音乐符号等其他形态的内容。
- 效能问题：需要下载整份文件，才能开始对文件做搜寻的动作。
- 扩充性、弹性、易读性均不佳。

为了解决以上问题，专家们使用 SGML 精简制作，并依照 HTML 的发展经验，产生出一套规则严谨，简单的描述数据语言 XML。XML 是在这样的背景下诞生的——是不是有一个更为中立的方式，让客户端自行决定如何消化、呈现从服务端所提供的信息呢？

XML 被广泛用作跨平台交互数据的形式，主要针对数据的内容，通过不同的格式化描述手段(XSLT、CSS 等)，可以完成最终的形式表达(生成对应的 HTML、PDF 或者其他文件格式)。XML 目的在于提供一个对信息能够做精准描述的机制，以弥补 HTML 太过于表现导向的特质。

**视频教学**：光盘/videos/13/ XML 简介.avi　　　　长度：6 分钟

XML(Extensible Markup Language)即可扩展标记语言，它与 HTML 一样，都是 SGML 的具体应用。与其他语言不同的是，XML 没有规定的标记，即用户可以根据需要自行创建符合 XML 规范的标记来描述 XML 文档要表达的内容。

XML 也是一种元标记语言，用于定义其他与特定领域有关的、语义的、结构化的标记语言的句法语言。

通过上面的介绍，我们了解到 XML 是一种可以自行创建的标记语言。使用户可以自行创建标记与元素，不仅用户能够理解文档的内容，而且具有相当高的灵活性。另外，XML 标记的主要特征是将内容与其表示法分离。

上面提到 XML 允许用户自行创建标记与元素，但需要遵循文档规范。那么如何保证 XML 的文档规范呢？很简单，即让 XML 遵循一定的文档结构。下面会详细介绍 XML 的文档结构。

13.2　创建一个标准的XML文档

我是一个购物爱好者，看到比较漂亮而且实惠的商品我都会马上订购。对经济并不富裕的我来说，总是偏爱于那些打折的商品，因此我时刻关注某些商品的打折价码。不知道大家仔细观察过没，在街道两边的商品每隔一段时间打折价都不相同，当时我就在想这么多商品，每件商品的打折都不相同，一个一个地标记岂不是太麻烦？现在我们可以使用 XML 文档来标记这些商品的打折价。

下面介绍如何使数据更易扩展和更新。

**视频教学**：光盘/videos/13/ XML 文档的结构.avi　　长度：17 分钟

13.2.1　基础知识——XML文档的结构

一个标准的 XML 文档结构，由 XML 的声明、XML 标记和元素、元素的属性、字符实体、CDATA 段、注释以及处理指令组成。下面详细介绍 XML 文档的组成部分，首先来看 XML 的声明。

1. XML的声明

XML 的声明是以<?xml 开始，以?>结束。例如：

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
```

在上述代码中，位于<? ... ?>中的各个参数的含义是什么？

- version：XML 的版本号。通常情况下，Web 支持的是 XML 1.0 版本。
- encoding：此 XML 文档的编码格式。常见的编码格式有 UTF-8 和 GB2312 两种。在 XML 文档中，默认的编码格式是 UTF-8。
- standalone：说明使用的是 DTD 形式，它有两个值 yes 或 no。XML 文档的默认值是 no，说明需要引用外部实体，而 yes 则说明所有必需的实体声明都包含在本文档中。

2. XML的标记与元素

众所周知，在 HTML 语言中，我们使用预定义标记来完成文档。例如，我们做了一位明星的履历，代码如下：

```
<dt>明星信息
<dd>我最喜爱的明星信息
  <ul>
    <li>A 明星</li>
    <li>男</li>
    <li>1.78m</li>
    <li>篮球、唱歌</li>
    <li>眼泪，是成长的代价。如哪天，你的泪不再流了，想说，那不是你长大了，而是，你的心已经死了</li>
  </ul>
```

在上述代码中，我们使用的是 HTML 标记语言，而且只能使用预定义标记来完成。但 XML 文档则不同，我们可以自定义标记。同样是定义明星的履历，可以这样写，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<明星>
  <名称>A 明星</名称>
  <性别>男</性别>
  <身高>1.78m</身高>
  <爱好>篮球、唱歌</爱好>
  <格言>眼泪，是成长的代价。如哪天，你的泪不再流了，想说，那不是你长大了，而是，你的心已经
死了</格言>
</明星>
```

从上述代码可以看出，XML 的标记是由<>来定义的。左边以尖括号<开始，右边以尖括号>结束，中间含有数据的标记称为非空标记。在上述 XML 文档中，<名称>和</名称>都是标记，<名称>、<性别>等称为开始标记，</名称>、</性别>等称为结束标记。



在 XML 文档中，标记都是成双成对出现的，有开始标记就少不了结束标记。

你可能会疑惑，既然有非空标记，那么是不是有空标记呢？答案是肯定的。

在 XML 文档中，标记可以分为非空标记和空标记。

下面来看一下空标记是如何定义的。

空标记是以尖括号<开始，以尖括号/>结束的标记。由于空标记不包含任何内容，因此空标记没有开始标记和结束标记，但空标记中可以包含属性。下面我们使用空标记来描述一下明星的履历，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<明星 名称="A 明星" 性别="男" 身高="1.78m" 爱好="篮球、唱歌" 格言="眼泪，是成长的代价" />
```

在上述代码中，名称、性别、身高、爱好等均是标记明星的属性。



该文档中提到的属性，在下面的章节中会详细介绍。

通过上面对标记的介绍，我们了解了如何定义标记，下面介绍一下 XML 文档中的元素。

元素是 XML 文档中最重要的构件，用来描述此文档所包含的数据。在 XML 文档中有且只有一个根元素，其他元素都是在根元素内部以树状结构显示的。

在 XML 文档中，元素可以分为非空元素和空元素两种类型。首先我们来看一下什么是非空元素。

所谓非空元素即是由开始标记、结束标记以及两标记之间的数据构成，开始标记和结束标记用来描述标记之间的数据，标记之间的数据称为元素的值。下面来看一下非空元素的语法格式。

```
<开始标记>描述的数据(元素的值)</结束标记>
```

在前面我们学习了空标记，其实空标记和空元素的格式是一样的。在 XML 解析器中，程序对空元素和空标记的处理方法是相同的，因此两者的作用是等价的。下面来看一下空元素的

定义。

所谓空元素就是不包含任何内容的元素，其语法格式如下：

```
<开始标记></结束标记>
```

空元素的语法格式还可以这样写：

```
<开始标记 />
```

为了更好地理解 XML 文档的元素，我们通过一个简单的 XML 文档来说明。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<用户>
  <编号 ID="123" ></编号>
  <用户名 >dcy</用户名>
  <身高 >1.67m</身高>
  <爱好 >爱的供养</爱好>
</用户>
```

在该文档中，“用户”是文档的根元素，根元素内部有 4 个子元素，分别是：编号、用户名、身高和爱好。其中<编号 ID="123" ></编号>是一个空元素，而其他 3 个子元素均是非空元素。当然，<编号 ID="123" ></编号>空元素还可以这样写，代码如下：

```
  <编号 ID="123"  />
```

瞧，这样是不是简化了 XML 文档呢。

俗话说“没有规矩不成方圆”，在任何编程语言中都有特定的语言规范，同样在 XML 文档中为了更加准确地描述数据，也有一套命名规则。下面我们来看一下如何对 XML 标记和元素进行命名。

- 标记或元素名称必须以字母、下划线或者中文开头，不能以数字或者标点符号开头。例如<_name>、<name>、<名称_name>等命名都是正确的。
- 元素名中可以包含字母、数字及其他字符，例如<name>、<用户名>、<ab123>等。但并不是所有的软件都能很好地支持中文命名，所以应尽量使用英文命名。在 XML 文档中，不要使用冒号来命名，此字符会在 XML 命名空间用到。
- 元素名称不能以 XML 的任意形式开头(如 XML 或者 Xml 等)，例如<xmlBook>、<XMLbook>、<XmlBook>等。
- 名称中不能包含空格。例如<abc　abc>。
- 文档中标记必须对应，也就是说在 XML 文档中只要有开始标记，就必须有结束标记，但空标记可以单独存在。
- 标记区分大小写。例如<name>和<Name>是两个完全不同的标记。

元素标记必须合理进行嵌套。

3. 元素的属性

XML 属性即是将一些额外的信息附加到元素上，从而使文档对元素数据描述更加具体。如果用户不喜欢通过子元素来描述元素的一些特性，那么使用属性来存储会是一个不错的选择。属性一般在开始标记中声明，由属性名和值构成。下面我们来看一下在非空元素中如何添加属性，语法如下：

```
<标记名 属性列表>描述的数据 </标记名>
```

在空元素中添加属性，语法如下：

```
<标记名 属性列表> </标记名>或者<标记名 属性列表/>
```

下面我们使用属性来描述今天的公交车情况。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<公交车>
  <车 95 长="4m" 宽="3m" 数量="40 人" >拥挤</车 95>
  <车 K6 长="4m" 宽="3m" 数量="20 人" > </车 K6>
</公交车>
```

在非空子元素“车 95”中，属性长、宽和数量描述了此路车的拥挤状况。而在空子元素“车 K6”中，属性长、宽和数量描述了此路车的宽松状况。

在 XML 中，标记和元素有命名规则，那么属性是不是也需要遵循一些命名规范呢？下面我们就来看一下。

- 属性名的命名规则和元素的命名规则相同，可以由字母、数字、中文以及下划线组成，但必须以字母、中文或者下划线开头。
- 属性名同样区分大小写。
- 属性值必须使用单引号或者双引号。例如"name"和'name'描述的是相同的属性值。
- 当属性值中需要使用左尖括号<、右尖括号>、连接符号&、单引号'或者双引号"时，必须使用实体引用。



这里提到的实体引用，在下面的章节中将会详细介绍。

在 XML 文档中，使用子元素和属性都可以实现对数据描述信息的存储。下面我们使用子元素来描述一下该文档的数据信息，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<公交车>
  <车 95>
    <长>4m</长>
    <宽>3m</宽>
    <数量>40 人</数量>
  </车 95>
  <车 K6>
    <长>4m</长>
    <宽>3m</宽>
    <数量>20 人</数量>
  </车 K6>
</公交车>
```

在 XML 文档中，使用属性和子元素并没有太大分别，只是根据执行环境和自己的喜好来决定使用哪种方式而已。在文档中使用属性，有以下缺陷：

- 属性不能包含多个重数值。
- 属性不容易扩展。
- 属性不能够描述文档结构。

- 属性很难被程序代码处理。
- 属性值很难通过 DTD 进行测试。

综合以上几点来看，在文档中使用属性仍然有很大的局限性。由于 XML 可扩展性高，因此数据在 XML 中需要经常添加或者修改，如果将这些数据放入属性中存储，那么对数据的维护和更新都很麻烦。

4. 字符实体

字符和实体引用可以向 XML 文档中引入其他信息，而不需要在文档中输入，例如在 XML 文档中会遇到大于号＞、小于号＜以及连接符号＆等，如果你想要将这些字符在 XML 文档中显示出来，那么你就要考虑实体引用了。

实体引用是以&开始，以;结尾的引用。

实体引用的作用是，当字符、数据中需要使用到这些特殊符号时，可以采用它的实体引用来代替。下面我们来看一下哪些特殊符号可以使用实体引用，如表 13-1 所示。

表 13-1　XML的实体引用

实体引用	特殊字符	含　义
<	<	小于号
>	>	大于号
&	&	连接符
'	'	单引号
"	"	双引号

看了表 13-1 中 XML 的实体引用，是不是感觉不可思议。下面我们通过一个例子来说明，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<生活>
  <名称>
    &lt;老男孩&gt;
  </名称>
  <样貌>
    &apos;生活像一把无情刻刀，&apos;    &amp;
    &quot;改变了我们模样&quot;
  </样貌>
</生活>
```

在上述文档中，我们使用了<、>、&、'、"等字符，程序运行时，解析器会将这些特殊字符解析成<、>、&、'和"。下面我们来看一下运行结果，是否与我们推测的一致，如图 13-1 所示。

5. CDATA段

上面的字符和实体引用很容易理解，但是这个 CDATA 看上去却很难。不是有这样一句话吗：世上无难事，只怕有心人。那么只要我们认真对待 CDATA，也就万事无忧了。

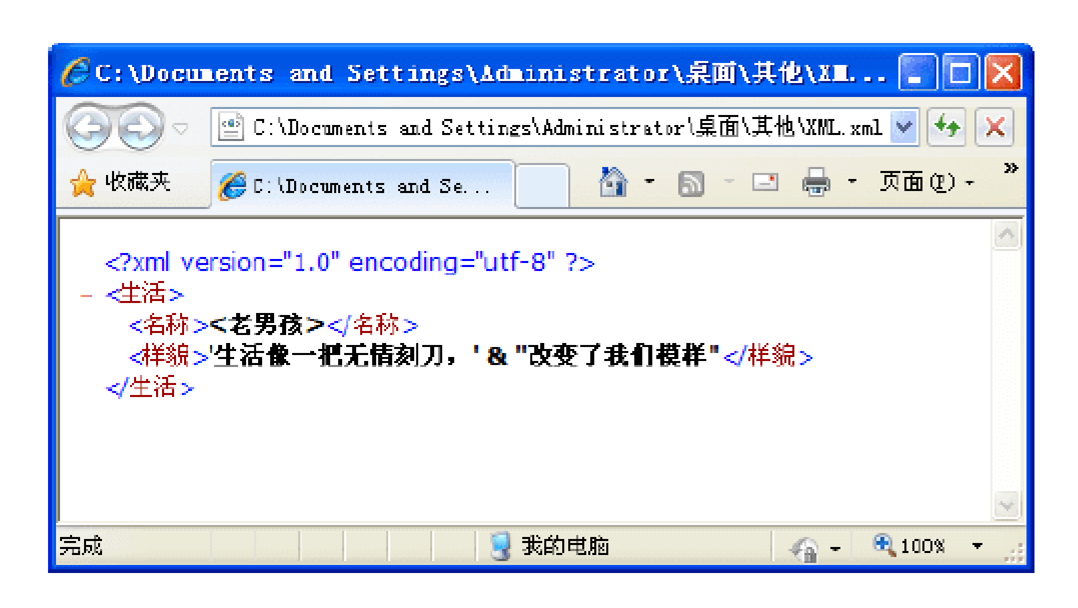


图 13-1　采用实体引用代替的效果

CDATA 区用来包含文本的方法，通常用于建立代码的脚本，例如 JavaScript 脚本。放在 CDATA 区的内容会被 XML 解析器忽略，然后被 XML 处理程序当作字符数据来处理。下面我们来看一下 CDATA 的语法格式。

```
<![CDATA[content]]>
```

了解了 CDATA 的语法格式，我们通过一个小例子来加深一下印象。下面有一段代码：

```
<?xml version="1.0" encoding="utf-8" ?>
<company>
  <employee>
    <name>dcy</name>
    <bumen>.NET 编程</bumen>
    <zhiwei>公司小员工</zhiwei>
    <hoppy>看电视</hoppy>
    <hate>自私自利</hate>
  </employee>
</company>
```

运行程序，执行结果如图 13-2 所示。

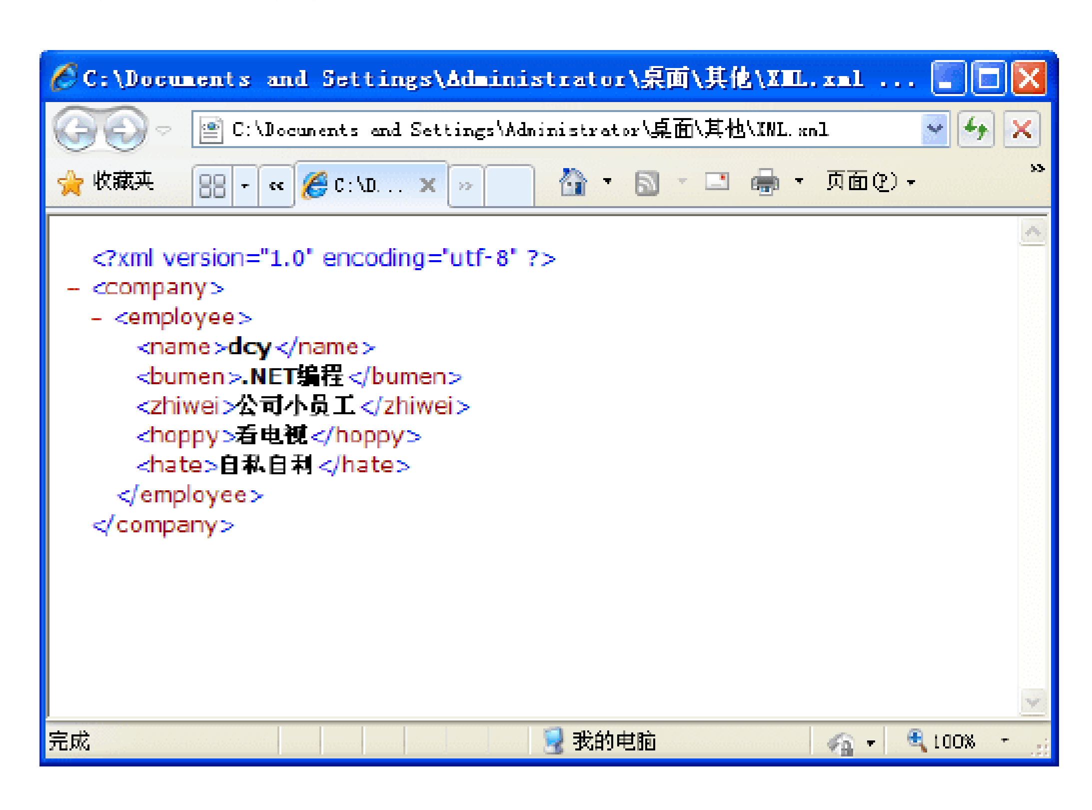


图 13-2　正常情况下显示的效果

在图 13-2 中，如果不想让 XML 解析器解析子元素中的属性 hoppy 和 hate，那么最理想的做法就是将这两个属性放到 CDATA 区，代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<company>
  <employee>
    <name>dcy</name>
```

```
    <bumen>.NET 编程</bumen>
    <zhiwei>公司小员工</zhiwei>
    <![CDATA[
    <hoppy>看电视</hoppy>
    <hate>自私自利</hate>
    ]]>
  </employee>
</company>
```

保存修改好的代码，然后运行程序，结果如图 13-3 所示。

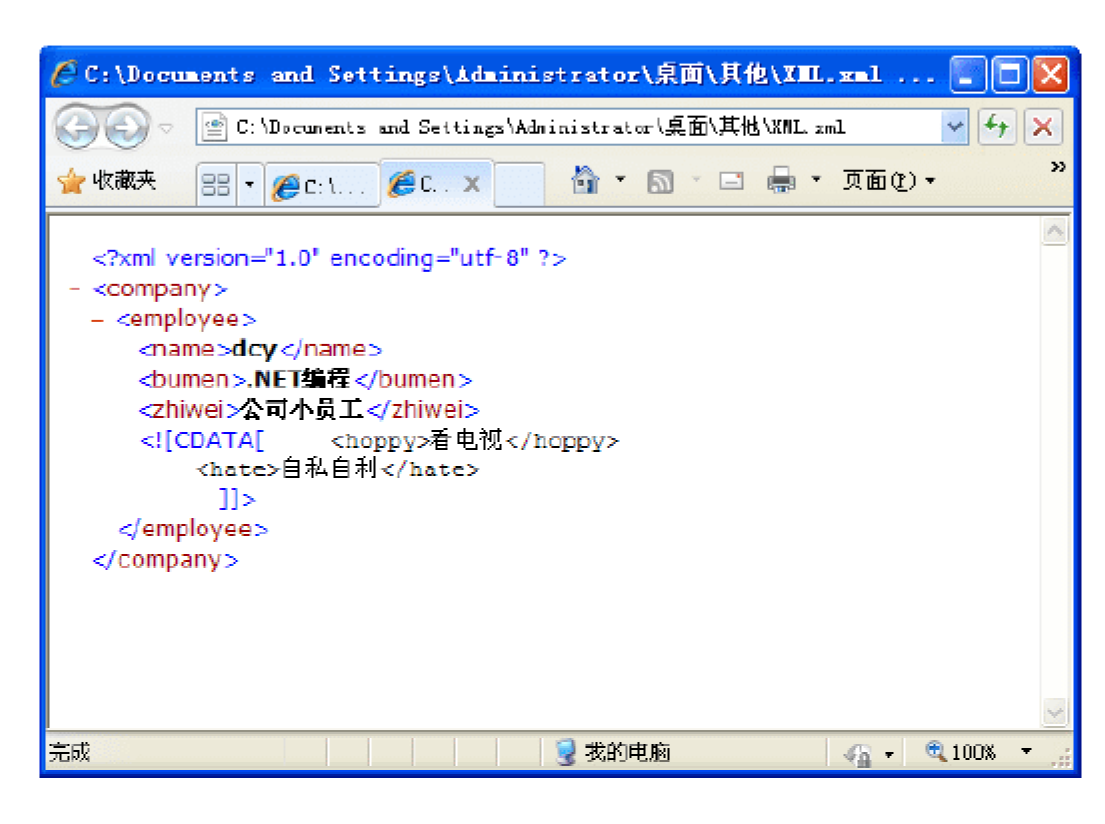


图 13-3　使用CDATA区的效果

前面我们提到 XML 处理程序会将 CDATA 区中的文本内容作为字符数据来处理，因此当文本文档大量使用特殊符号时，我们就可以这样写代码，例如：

```
<?xml version="1.0" encoding="utf-8" ?>
<歌词>
  <![CDATA[
  <名称>
    <爱的供养>
  </名称>
  <语句>
    '我用尽一生一世来将你供养，只盼望能停住你流转的目光'&
    "请赐予我无限爱与被爱的力量，让我能安心在菩提树下静静地思量"
  </语句>
  ]]>
</歌词>
```

这样是不是比那些通篇文档都用实体引用代替简便多了。

6. 注释

XML 注释是对文档结构或者内容的解释，由于它不属于 XML 文档的内容，因此 XML 解析器不会处理。注释以<!--开始，以-->结束，其语法格式如下：

```
<!--需要注释的内容-->
```

注释主要可以用在文档的序言部分、文档的文本内容中以及文档之后。但是-->字符不能用于注释的内部，由于解析器一旦碰到-->就会当作一个注释的结束，而后面的内容则被作为普通的 XML 文档处理。其他合乎规范的 XML 字符均可出现在注释的内部。

7. 处理指令

XML 文档通过处理指令(Processing Instruction，PI)可以包含用于特定应用的指令。一个处理指令通过<?和?>来指定，紧接<?后面的是处理的指令名，其次属性值对。你可能会在 XHTML 文档中遇到这样的语句：

```
<? robots index="yes" follow="no" ?>
```

在上述语句中，其中 robots 是处理指令的名称，指的是搜索引擎是否检索以及如何检索一个页面的指示。这条处理指令有两个属性，分别是 index 和 follow，它们的值分别是 yes 和 no。另外，我们在 XML 文档中经常会遇到这样的代码：

```
<?xml version="1.0" encoding="utf-8" ?>
```

上面这段代码从形式上看，也是一个处理指令，它的作用是对 XML 文档进行标识。

13.2.2 实例描述

我朋友家是开超市的，我去那里帮忙，恰巧碰到厂家来送货，朋友交代我将这些商品的价格记录下来，并且将记录下来的商品进行分类。于是我一直在那里记录，不久我就被搞晕了，这样记录连我自己都看不太明白，更别提别人了。突然间我想能不能使用 XML 文档来将这些商品作总体分类呢，然后将属于某一类的商品作为其类的子元素？

因为 XML 的可扩展性很高，并且很容易维护和修改。于是我就马上行动起来，很快就将这些商品分类成功了。我不得不惊叹学以致用真的很重要，下面来看一下我的实现方案。

13.2.3 实例应用

【例 13-1】创建一个标准的 XML 文档。

(1) 创建一个名称为 shangpin.xml 的文件。

(2) 在 shangpin.xml 文件中添加代码。

```
<?xml version="1.0" encoding="utf-8" ?>
<goods>
<!-- 我自己的商品-->
  <SkinCare>
    <anti_wrinkle>
      <SkinName>真奢华</SkinName>
      <SkinPrice>450 元</SkinPrice>
      <SkinEffect>去干纹</SkinEffect>
    </anti_wrinkle>
    <hydrating>
      <SkinName>dermare</SkinName>
      <SkinPrice>250 元</SkinPrice>
      <SkinEffect>双因子补水维护精华</SkinEffect>
    </hydrating>
    <whitening >
      <SkinName>美白</SkinName>
      <SkinPrice>270 元</SkinPrice>
```

```
      <SkinEffect>美白维护精华</SkinEffect>
    </whitening>
  </SkinCare>
  <shampoo>
    <repair >
      <shampooName>潘婷</shampooName>
      <shampooPrice>50 元</shampooPrice>
      <shampooEffect>修复干枯洗发露</shampooEffect>
    </repair>
    <Prevent>
     <![CDATA[
      <shampooName>迪彩</shampooName>
      <shampooPrice>30 元</shampooPrice>
      <shampooEffect>防干枯洗发露</shampooEffect>
       ]]>
    </Prevent>
  </shampoo>
</goods>
```

(3) 保存修改好的代码。

13.2.4　运行结果

运行程序，执行结果如图 13-4 所示。

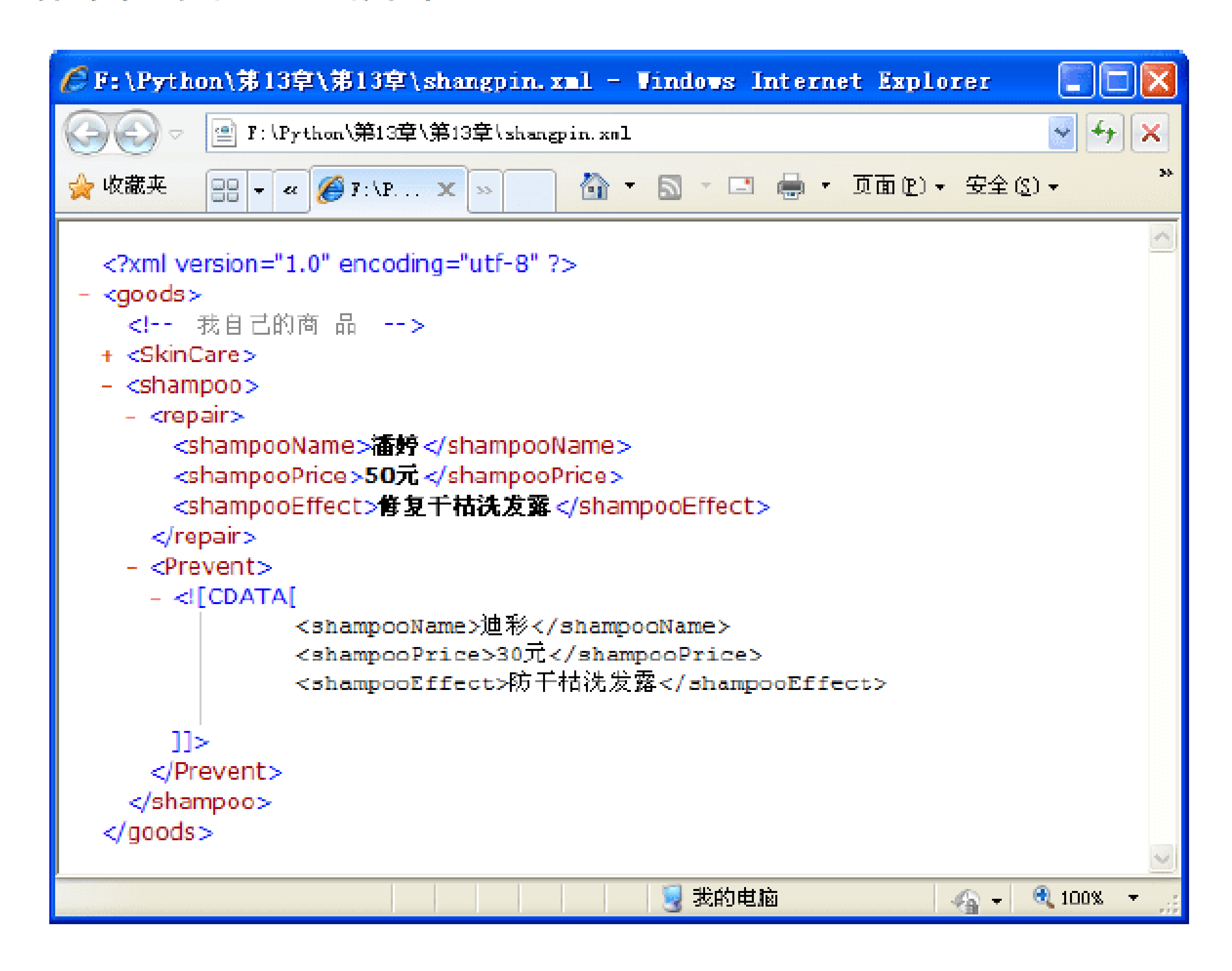


图 13-4　显示效果

13.2.5 实例分析



源码解析

在本实例中，使用商品 goods 作为此文档的根元素，以护肤品 SkinCare 和洗发露 shampoo 作为此根元素的子元素。接着 anti_wrinkle、hydrating 和 whitening 则是子元素 SkinCare 下的嵌套子元素，而 repair 和 Prevent 则是在子元素 shampoo 下的商品。另外我们还使用了<!-- -->注释以及<![CDATA[]]>段。瞧，这样算不算是一个标准的 XML 文档呢？

13.3 读取XML文档节点下的数据

在 Python 中如何对 XML 的数据进行读取呢？接着往下看，在该节中将会介绍使用 SAX 读取 XML 中的数据。



视频教学：光盘/videos/13/读取 XML 文档节点下的数据.avi　　长度：3 分钟

13.3.1 基础知识——SAX介绍

SAX(XML的简单API)全称Simple API for XML，它是XML语法分析器的公用语法分析器接口，同时也是一个软件包。它允许应用程序作者编写使用XML语法分析器的应用程序，但是它却独立于所使用的语法分析器。其原理是对文档进行顺序扫描，当扫描到文档(document)开始与结束、元素(element)开始与结束、文档(document)结束等地方时，立即通知事件处理函数，由事件处理函数做相应动作，然后继续同样的扫描，直至文档结束。

目前，新的接口标准是 SAX2，该接口包含了对 XML 名字命名空间的支持。在 Python 中，SAX2 实现的是 xml.sax 模块，在此模块中实现了 4 种类型的接口，如表 13-2 所示。

表 13-2 xml.sax模块中支持的SAX2 接口

接　口	说　明
ContentHandler	内容处理接口
DTDHandler	DTD 定义处理接口
EntityResolver	实体解析器处理接口
ErrorHandler	错误信息处理接口

其中，ContentHandler 接口封装了一些事件处理的方法，当 XML 解析器开始解析 XML 文档时，它会遇到某些特殊的事件，比如文档的开头和结束、元素开头和结束，以及元素中的字符数据等事件。当遇到这些事件时，XML 解析器会调用 ContentHandler 接口中相应的方法来响应该事件，例如 void startDocument()、void endDocument()等。

DTDHandler 用于接收基本的 DTD 相关事件的通知，该接口仅包括 DTD 事件的注释和未解析的实体声明部分。SAX 解析器可按任何顺序报告这些事件，而不管声明注释和未解析实体时所采用的顺序。但是，必须在文档处理程序的 startDocument()事件之后，且在第一个 startElement()事件之前报告所有的 DTD 事件。

EntityResolver 接口是用于解析实体的基本接口。

ErrorHandler 接口是 SAX 错误处理程序的基本接口。如果 SAX 应用程序需要实现自定义的错误处理，则它必须实现此接口，然后解析器将通过此接口报告所有的错误和警告。

13.3.2　基础知识——SAX处理的组成部分

xml.sax 模块由 4 种主要的处理器接口组成。每种接口都会在特定的时候被解析触发，例如 ContentHandler 会在系统读取文件特定内容的时候触发。这些处理器可以在一个对象中实现，也可以分布在多个对象中，这些对象的实现需要继承 xml.sax.handler 模块中的类。下面将具体介绍这 4 类处理器以及 XMLReader 对象的使用方法。

1. ContentHandler接口

ContentHandler 接口是在使用 SAX 技术时用得最多的处理器对象，此类提供了丰富的方法用以处理 XML 文档数据。在 ContentaHandler 基类中主要包含的方法如表 13-3 所示。

表 13-3　ContentHandler基类中常用的方法

方　法	说　明
startDocument()	文档处理开始的时候调用
endDocument()	文档处理结束的时候调用
startElement(name,attrs)	遇到开始元素时触发，name 是元素名，attrs 是元素属性字典
endElement(name)	遇到结束元素时触发，name 是元素名
startElementNS(name,qname,attrs)	处理名字空间，遇到开始元素时触发，name 是元素名(一个元组)，包含 URI 和本地名，如 namespace:title 返回('namespace', 'title')。qname 是从 XML 中标识的原始元素名，attrs 是元素属性字典
endElementNS(name,qname)	处理名字空间，遇到结尾元素时触发，name 和 qname 的含义同上
characters(content)	遇到字符数据时触发
ignorableWhitespace()	可以忽略空白字符处理的时候调用
processingInstruction(target,data)	收到处理指令的时候调用
skippedEntity(name)	在跳过实体时触发

通过表 13-2 中介绍的方法，使得 SAX 技术在处理 XML 文档的时候能够做到快速和完整。大多数人可能会对 startElement 方法感兴趣，因为在数据解析中遇到的每个元素，这个方法所注册的函数都会被调用。该方法有两个参数，分别是标签名称和属性，通过判断名称和属性值来实现 XML 文档所需要的操作。接下来我们来看一个例子，代码如下：

```
from xml.sax import*
class myHandler(ContentHandler):
```

```
    def startDocument(self):                       #开始处理文档时调用
        print '-------开始处理 XML 文档------'
        print 'name\tprice\taffect'
        print '--------------------------'
    def endDocument(self):                         #处理文档结束时调用
        print '-------策处理结束 XML 文档-'
    def startElement(self,name,attrs):             #元素开始的时候调用
        if name=='attr':
            print '%s\t%s\t%s'%(attrs['name'],attrs['price'],attrs['affect'])
parser = make_parser()                             #返回一个 XML 解析对象
parser.setContentHandler(myHandler())              #为内容处理器设置值
#-----------------解析 XML 数据
data='<goods><attr name="zhenshehua" price="450" affect="quganwen"/><attr
name="dermare" price="250" affect="bushuijinghua"/><attr name="meibai"
price="270" affect="meibaijinghua"/></goods>'
import StringIO                                    #导入 StringIO 模块
parser.parse(StringIO.StringIO(data))              #对 XML 数据进行分析
```

在上述代码中，使用 from xml.sax import*方式将模块中的方法和对象导入，接着定义了一个 myHandler 类，使其继承 ContentHandler 类。在该类中定义了 3 个方法，分别是 startDocument、endDocument 和 startElement。其中，startDocument 方法在该类中所有方法执行之前执行，而 endDocument 方法则在该类中所有方法之后执行。接着使用 xml.sax 模块下的 make_parser 方法返回一个 XML 的解析对象，然后将 myHandler 类的实例对象传入到解析对象处理器 setContentHandler 中，最后调用 parse 方法对 XML 数据进行分析。下面看一下执行的结果。

```
>>>
-------开始处理 XML 文档------
name    price   affect
--------------------------
zhenshehua  450 quganwen
dermare 250 bushuijinghua
meibai  270 meibaijinghua
-------策处理结束 XML 文档-
>>>
```

接下来将介绍的是 DTDHandler 接口。

2. DTDHandler接口

当你遇到需要处理有关 DTD 定义的内容时，可以使用 DTDHandler 对象，该对象提供了两个方法，分别是 notationDecl 和 unparsedEntityDecl。其中，notationDecl 方法主要用来处理 DTD 定义中的符号定义，而 unparsedEntityDecl 方法主要用来对实体定义进行解析。已解析的对象可以通过 setDTDHandler 方法来传入处理函数，下面一段代码就是如何处理 DTD 定义的框架。

```
from xml.sax import*
class myHandler(DTDHandler):
   pass
parser = make_parser()
myhandler=myHandler()
parser.setDTDHandler(myhandler)
```

在上面的代码中，创建了一个 myHandler 类，并且继承自 DTDHandler 类，接着调用

解析对象的 setDTDHandler 方法来处理该 DTD 对象。

3. EntityResolver接口

前面我们提到，EntityResolver 接口是为实体参考解析所设置的接口。也就是说，当解析器遇到外部实体参考的时候就会调用该对象中的 resolveEntity 方法来处理。接下来我们来看一下外部处理实体参考的框架，代码如下：

```
from xml.sax import*
class myHandler(EntityResolver):
   pass
parser = make_parser()
myhandler=myHandler()
parser.setEntityResolver(myhandler)
```

在上述代码中，首先导入 xml.sax 模块，接着创建一个 myHandler 类，并使其继承 EntityResolver，然后调用 parser 解析对象的 setEntityResolver 方法来设置该外部实体的参考处理对象。

4. ErrorHandler接口

ErrorHandler 接口使应用程序在处理 XML 文档时能够实现错误处理。该处理错误信息的框架如下：

```
from xml.sax import*
class myHandler(ErrorHandler):
   pass
parser = make_parser()
myhandler=myHandler()
parser.setErrorHandler(myhandler)
```

在上述代码中，我们定义了一个 myHandler 类，使其继承 ErrorHandler 类，接着调用 make_parser 方法生成一个解析对象，然后调用 setErrorHandler 方法来设置错误处理函数。

在 Java 中，错误是分级别的，这里也一样。ErrorHandler 类中提供的错误方法共有 3 个，分别是 warning、error 和 fatalError。其中，warning 方法主要处理那些在解析 XML 数据时所出现的微小错误，当遇到这类错误时，默认情况下会打印出错误信息，然后继续处理后面的数据流。error 方法主要处理比较严重但可以恢复的错误，当遇到该错误时，如果不抛出异常，则继续解析后面的数据，但不能保证后面解析的数据一定是正确的。fatalError 方法处理那些非常严重而且不能恢复的错误，当遇到这类错误时，解析就会停止。

warning、error 和 fatalError 方法都只能接收一个参数，即 SAXException 异常类的实例。也就是说，如果想终止对 XML 数据的解析，直接触发 SAXException 异常即可。

5. XMLReader对象

上面提到的 4 个接口都需要将这些实现的接口注册在一个 SAX 的解析对象上。也就是说，需要使用 xml.sax 的 make_parser 方法生成一个 XMLReader 对象，接着通过(例如 setErrorHandler())方法来设置相应的接口。如果想要从当前解析对象中获取已经注册的处理器接口，只需将上述方法中的 set 换成 get，其代码格式如下：

```
handler=parser.getErrorHandler()
```

在该接口中最重要的方法是 parse，调用该方法将开始解析 XML 数据并产生相应的事件信息，包括内容信息、DTD 信息等。该方法接收一个参数，该参数可以是文件名、文件的 URL 或者文件对象，甚至是 InputSource 对象。下面看一下 XMLReader 接口提供了哪些方法，如表 13-4 所示。

表 13-4　XMLReader接口中常用的方法

方　法	说　明
Parse	解析 XML 数据并产生 SAX 事件信息
getContentHandler	获取 XML 数据内容的处理函数
setContentHandler	设置 XML 数据内容的处理函数
getDTDHandler	获取 XML 数据的 DTD 定义的处理函数
setDTDHandler	设置 XML 数据的 DTD 定义的处理函数
getEntityResolver	获取 XML 数据的外部实体参考的处理函数
setEntityResolver	设置 XML 数据的外部实体参考的处理函数
getErrorHandler	获取 XML 数据的错误处理函数
setErrorHandler	设置 XML 数据的错误处理函数
setLocale	设置警告和错误信息的本地化
getFeature	获取特性数据
setFeature	设置特性数据
getProperty	获取属性数据
setProperty	设置属性数据

针对表 13-4 中的 getContentHandler 方法，我们来看一个例子，代码如下：

```
from xml.sax import*
class myHandler(ContentHandler):
    def startDocument(self):                    #开始处理文档时调用
        print '-------start Document------'
        print 'name\tprice\taffect'
        print '--------------------------'
    def endDocument(self):                      #处理文档结束时调用
        print '----------end Document----------'
    def startElement(self,name,attrs):
        if name=='attr':
            print '%s\t%s\t%s'%(attrs['name'],attrs['price'],attrs['affect'])
parser = make_parser()
parser.setContentHandler(myHandler())
handler=parser.getContentHandler()
print '获取的注册处理器接口是：',handler
data='<goods><attr name="My Dream" price="wu jia" affect="qu bei jing"/><attr
name="dermare" price="250" affect="wo de ai"/><attr name="like" price="throw"
affect="ni"/></goods>'
import StringIO
parser.parse(StringIO.StringIO(data))
```

在上述代码中，首先使用 startDocument、endDocument 以及 startElement 方法来处理文档，接着使用 xml.sax 模块下的 make_parser 方法返回一个 XML 的解析对象，然后将 myHandler 类

的实例对象传入到解析对象处理器 setContentHandler 中，最后使用 parser 的 getContentHandler 方法来获取已经注册的处理器接口。执行结果如下：

```
>>>
获取的注册处理器接口是： <__main__.myHandler instance at 0x011DB198>
-------start Document------
name    price   affect
--------------------------
My Dream    wu jia  qu bei jing
dermare 250 wo de ai
like    throw   ni
----------end Document----------
>>>
```

13.3.3　实例描述

有一句话这样说：如果你不逼自己一把，你永远不知道你自己的能力有多大。就是这句话给我了信心，让我有独揽大项目的勇气。在该项目中，使用了一个 XML 文档来存储大量的数据，因此从 XML 文档中寻找我想要的标签就成了最大的难题，面对大量的数据，头昏目眩。学了使用 SAX 处理 XML 文档数据，给了我一些灵感，我可以开发一个小程序，在该程序中只需要输入你想要查询的标签就可以查询该标签下的属性值。下面来看一下我的实现方案。

13.3.4　实例应用

【例 13-2】 读取 XML 文档节点下的数据。

(1) 创建一个名称为 xmlRead.py 的文件。

(2) 在 xmlRead.py 文件中添加代码。

```
import string
from xml.sax import*
class QuotationHandler(ContentHandler):
    def __init__(self):
        self.string=''
    def startDocument(self):                    #开始处理文档时调用
        print '-------开始处理 XML 文档------'
        print 'name\tprice\taffect'
        print '--------------------------'
    def endDocument(self):                      #处理文档结束时调用
        print '-------处理结束 XML 文档-'
    nameStr=raw_input('请输入你想查看 XML 文档中的标签(title): ')
    def startElement(self, nameStr, attrs):
        print '-----------Start Element-----------'
        if nameStr == 'title':
            print '------标签为 title 下的数据-------'
            print attrs['name'],attrs['offer_id'],attrs['mobile_url']
    def characters(self, ch):
        self.string = self.string + ch
if __name__ == '__main__':
    try:
```

```
        parser =make_parser()
        handler = QuotationHandler()
        parser.setContentHandler(handler)
        parser.parse("sample.xml")
    except:
        import traceback
        traceback.print_exc()
```

(3) 保存修改好的代码。

13.3.5 运行结果

运行程序，当输入标签为 title 时，执行结果如下：

```
>>>
请输入你想查看 XML 文档中的标签(title): title
-------开始处理 XML 文档------
name    price   affect
--------------------------
-----------Start Element-----------
-----------Start Element-----------
------标签为 title 下的数据-------
有一种情调叫做浪漫 2 http://localhost:8080
-----------Start Element-----------
------标签为 title 下的数据-------
痞子英雄中的电脑高手好强 3 192.168.1.1
-----------Start Element-----------
------标签为 title 下的数据-------
如何根据手机定位 4 192.168.1.2
-----------Start Element-----------
------标签为 title 下的数据-------
Java 编程 5 192.168.1.3
-----------Start Element-----------
------标签为 title 下的数据-------
如何根据手机定位 6 192.168.1.4
-------处理结束 XML 文档-
>>>
```

13.3.6　实例分析



源码解析

在本实例中，创建了一个 QuotationHandler 类，并使之继承 ContentHandler 类，接着使用了 startDocument、endDocument、startElement 和 characters 四个方法来处理文档，之后调用 make_parser 方法返回一个解析器对象 parser，最后将 myHandler 类的实例对象传入到解析对象处理器 setContentHandler 中。从实例运行结果可以看出，程序每执行一次都需要调用 startElement 方法。

13.4　从XML文件中读取数据库配置

在使用.NET 开发大型网站或者系统时，往往会将数据库的配置信息保存到一个 XML 文件中，在该 XML 文件中有一个根节点 configuration，其次是子节点 appSettings。在该子节点下还有很多子节点，例如名称是 add 的子节点有两个属性：key 和 value。在程序中使用 ConfigurationSettings.AppSettings[key]的方式就能将 value 的值读取。那么在 Python 中如何读取数据库的配置文件呢？XML 配置文件有特殊的规则吗？

**视频教学：**光盘/videos/13/DOM.avi　　　　**长度：**13 分钟

13.4.1　基础知识——DOM介绍

DOM(Document Object Model，文档对象模型)以一种独立于平台及语言的方式来访问与修改一个文档的内容和结构。由于DOM的设计是以对象管理组织(Object Management Group，OMG)的规约为基础的，因此可以用于任何编程语言。最初，人们认为它是一种让JavaScript在浏览器间可移植的方法，但现在DOM的应用已经远远超出这个范围。DOM技术使得用户页面可以动态地变化，例如可以动态地显示或隐藏一个元素，改变它们的属性，增加一个元素等，DOM技术使得页面的交互性有很大的增强。

DOM 实际上是以面向对象方式描述的文档模型。DOM 定义了表示和修改文档所需的对象，这些对象的行为和属性，以及这些对象之间的关系。也可以把 DOM 认为是页面上数据和结构的一个树形表示，不过需要注意的是页面可能并不是以这种树的方式具体呈现。

DOM 技术在发展过程中，不免有新模块的加入和旧模块的废除，表 13-5 列出了在 DOM 技术中主要支持的特性，这些特性在程序中经常用到。

根据 W3C DOM 规范，DOM 是 HTML 与 XML 的应用编程接口(API)，DOM 将整个页面映射为一个由层次节点组成的文件，有 1 级、2 级、3 级共 3 个级别。由于 DOM 的技术是由一系列规范组成的，所以内容很丰富。从操作上来说，DOM 提供接口将 XML 文档显示成 DOM 节点对象的分层树结构。在形成这种文档结构中，有些节点下面可以有不同类型的子节点，而

其他一些则是子节点。其中 DOM 的文档节点和根节点是对应的，根节点下面可以包括子元素节点和属性节点，而子元素节点又可以包括子元素节点以及文本节点，这样一个 DOM 树结构的图就显示出来了，如图 13-5 所示。

表 13-5　DOM技术支持的特性

特　性	说　明
Core	此特性包含处理结构的 XML 文档所需的基础结构，但不包含任何 DTD 定义的信息，也不包含外部实体参考、处理指令等相关信息。在该特性中，提供了众多的接口来处理文档数据
XML	该特性包含 XML 相关的信息，包括之前没有规范的实体参考和定义等，也包括 CDATA 段和处理指令
Events	该特性的各个实现之间的差别比较大，而且已经分成了很多子特性。各种实现虽然千差万别，但都支持最基本的 Events 特性，这些事件中，包括面向用户的操作接口和对文档进行修改操作的接口
Range	该特性为转换为一系列 DOM 节点的文档数据处理的提供接口
Traversal	该特性支持应用程序在文档节点之间进行移动，当生成树状结构后，可以方便地实现对父节点、子节点的遍历
Views	为一个文档数据提供了多种不同类型的显示
CSS	该特性为文档数据提供了对 CSS2 规范的访问接口。这个特性一般用在 XML 数据的浏览器显示

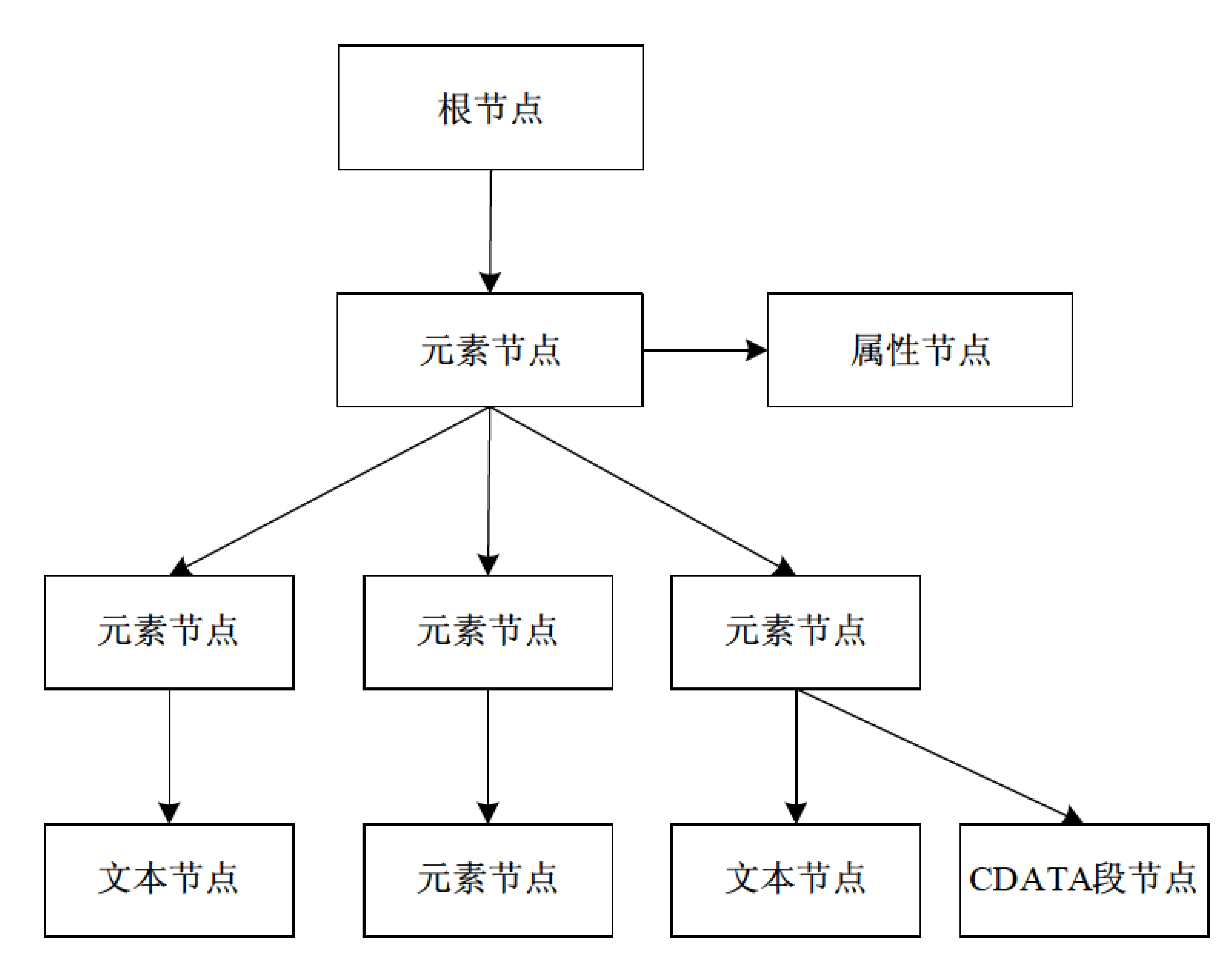


图 13-5　DOM树状结构

由于 DOM 的规范过程是分阶段的，所以形成了 DOM 的层次，现在广泛使用的是 DOM2。在 Python 标准库中支持 DOM 操作的模块是 xml.dom，主要支持的是 DOM2 标准，并且在该模块中提供了 getDOMImplementation 方法来检测标准库是否支持某种特性。

13.4.2　基础知识——xml.dom模块中的接口操作

当 XML 文档数据通过 xml.dom 模块中的方法解析为 DOM 树状结构后，便可以调用 DOM 规范中定义的方法和属性来访问 XML 数据，其中该方法是在 DOM 中规定的，而不是在 xml.dom 中。xml.dom 模块只是实现了这些操作 XML 数据的接口，表 13-6 列出了在 xml.dom 模块中实现的接口。

表 13-6　xml.dom模块中实现的接口

接　口	说　明
DOMImplemetation	应用程序判断 DOM 实现是否支持某些特性
Document	表示整个 XML 文档
Node	表示文档结构的单个节点
Element	是 Node 的子类，主要实现对元素节点的一些操作
Text	实现了 Node 接口，主要用来保存数据

上面我们了解了在 xml.dom 模块中可以实现的接口操作，接下来将介绍这些接口如何使用，首先来看 DOMImplemetation 接口。

1. DOMImplemetation接口

DOMImplemetation 接口使得应用程序可判断 DOM 实现是否支持某些特性，此时需要调用 hasFeature 方法来测试特性是否存在，如果存在，则返回 True，否则返回 False，其格式如下：

```
DOMImplementation().hasFeature('character','levels')
```

在上述语法中，其中 character 指的是 DOM 规范所支持的特性，levels 值是 DOM 规范的层次。接下来我们通过一个例子来说明 hasFeature 方法的使用，代码如下：

```
from xml.dom.minidom import DOMImplementation
implementation=DOMImplementation()
print implementation.hasFeature('Core','2.0')
print implementation.hasFeature('Events','2.0')
print implementation.hasFeature('Traversal','2.0')
print implementation.hasFeature('Views','2.0')
print implementation.hasFeature('CSS','2.0')
print implementation._features
```

在上述代码中，我们使用 from xml.dom.minidom import DOMImplementation 的方式将 DOMImplementation 接口导入，minidom 是 DOM 接口的一个轻量级实现。接着创建了一个 implementation 对象，然后调用该对象的 hasFeature 方法来判断现在版本的 DOM 实现是否支持该特定层次的特性，最后调用_features 属性将当前 DOM 接口实现所支持的所有特性打印输出。执行结果如下：

```
True
False
False
False
False
```

```
[('core', '1.0'), ('core', '2.0'), ('core', '3.0'), ('core', None), ('xml',
'1.0'), ('xml', '2.0'), ('xml', '3.0'), ('xml', None), ('ls-load', '3.0'),
('ls-load', None)]
```

2. Document接口

Document 接口用来表示整个 XML 文档。上面我们介绍了 DOMImplemetation 接口，这里我们可以使用该接口对象的 createDocument 和 createDocumentType 方法来生成 Document 接口对象，代码如下：

```
from xml.dom.minidom import DOMImplementation
implementation=DOMImplementation()
documentype=implementation.createDocumentType('dcy','','dcy')
document=implementation.createDocument('','dcys',documentype)
print document
```

在上述代码中，我们使用对象 implementation 的 createDocumentType 方法生成了一个文档定义对象，并作为参数传入到 createDocument 方法中。其执行结果如下：

```
<xml.dom.minidom.Document instance at 0x00B9B5A8>
```

这样就生成了一个 xml.dom.mindom.Document 接口的实例，很简单吧。

除了使用上面的方法来生成 Document 接口的实例外，还可以通过解析 XML 文档数据来生成。需要注意的是，在 minidom 模块中有 parse 和 parseString 两个方法，其中 parse 方法接收的参数可以是文件，也可以是文件对象，而 parseString 方法接收的参数则是字符串。接下来我们来看一段代码。

创建一个名称为 my.xml 的文件，并在文件中添加如下代码：

```
<dream>
<myDream quality='myDream'>
<say>you are the best one</say>
</myDream>
</dream>
```

接着创建一个.py 的文件，并在该文件中添加如下代码：

```
from xml.dom.minidom import parse,parseString
domW=parse('my.xml')
print '我传入的是文件：',domW
fle=open('my.xml','r')
domD=parse(fle)
print '我传入的是文件对象：',domD
data="""
<dream>
   <myDream quality='myDream'>
     <say>you are the best one</say>
   </myDream>
</dream>
"""
domS=parseString(data)
print '我传入的是字符串：',domS
```

在上述代码中，分别使用 parse 和 parseString 方法来对 XML 数据进行解析，这样会不会生成 Document 对象呢？执行结果如下：

```
>>>
```

```
我传入的是文件： <xml.dom.minidom.Document instance at 0x011D8A80>
我传入的是文件对象： <xml.dom.minidom.Document instance at 0x011DCDA0>
我传入的是字符串： <xml.dom.minidom.Document instance at 0x011E5148>
>>>
```

前面提到 Document 接口是整个文档结构的根，那么它的输出即为 XML 文档的输出，通过调用 toxml 方法可将整个文档输出。下面通过一个例子来说明 toxml 方法的使用，这里仍使用上面定义好的 my.xml 文件，代码如下：

```
from xml.dom.minidom import parse,parseString
domW=parse('my.xml')
print domW.toxml()
```

执行结果如下：

```
>>>
<?xml version="1.0" ?><dream>
<myDream quality="myDream">
<say>you are the best one</say>
</myDream>
</dream>
>>>
```

3. Node接口

Node 接口是文档对象模型的基本数据类型，表示构成文档结构的每个单节点。也就是说，在 DOM 中的结构都是 Node 对象，每个节点的类型通过 nodeType 来区别，不同节点对象有着不同的节点类型，接下来我们来看一个例子。

在该例中，使用的 XML 文件仍然是上面提到的 my.xml 文件。代码如下：

```
from xml.dom.minidom import *
dom=parse('my.xml')
domRoot=dom.documentElement
print domRoot
print domRoot.nodeType
print dom.ATTRIBUTE_NODE
```

在上述代码中，使用了 Document 接口中的 documentElement 属性来获得根元素节点，并赋值给 domRoot，然后通过 nodeType 属性来查看节点的类型。下面来看一下执行的结果。

```
<DOM Element: dream at 0xba7850>
1
2
```

从上述结果可以看出，通过调用 nodeType 得到的节点类型是 1，调用 ATTRIBUTE_NODE 属性得到的节点类型的值是 2。当然，在 Node 接口中还有其他的节点类型，如表 13-7 所示。

表 13-7　Node接口中支持的节点类型

节点类型	数值表示
ELEMENT_NODE	1
ATTRIBUTE_NODE	2
TEXT_NODE	3

CDATA_SECTION_NODE	4
ENTITY_REFERENCE_NODE	5
ENTITY_NODE	6
PROCESSING_INSTRUCTION_NODE	7
COMMENT_NODE	8
DOCUMENT_NODE	9
DOCUMENT_TYPE_NODE	10
DOCUMENT_FARGMENT_NODE	11
NOTATION_NODE	12

由于 XML 文档可以解析成树结构对象，因此这里的节点也就具有树的特性。可以通过 parentNode 属性来获取父节点，通过 childNodes 来获取子节点的集合。通过 firstChild 和 lastChild 分别可以获取第一个和最后一个子节点。另外，还可以通过 previousSibling 和 nextSibling 获得属于同一父节点下的前一个和后一个的子节点。接下来通过一个例子来说明，代码如下：

```
from xml.dom.minidom import *
dom=parse('my.xml')
domRoot=dom.documentElement
print 'domRooot 父节点',domRoot
childs=domRoot.childNodes
print '所有子节点',childs
mydream=domRoot.childNodes[1]
print '打印输出子节点',mydream
parent=mydream.parentNode
attr=mydream.attributes
print '获得父节点',parent
print 'parent 和 domRoot 对象是相同的',parent==domRoot
print 'domRoot 元素的第一个子节点：',domRoot.firstChild
print 'domRoot 元素的最后一个子节点：',domRoot.lastChild
print '获取同一父节点下的前一个子节点：',mydream.previousSibling
print '获取同一父节点下的后一个子节点：',mydream.nextSibling
print '获取 Node 对象的所有属性值：',attr.items()
```

在上述代码中，通过使用 childNodes 属性获得了根节点下的所有子节点对象。接着调用 firstChild 和 lastChild 分别得到 domRoot 根元素节点的第一个子节点和最后一个子节点。之后通过使用 previousSibling 和 nextSibling 分别得到 mydream 元素节点的前一个和后一个兄弟节点，最后通过 attributes 属性来获取 Node 对象的所有属性值。执行结果如下：

```
>>>
domRooot 父节点 <DOM Element: dream at 0x11dc918>
所有子节点 [<DOM Text node "
">, <DOM Element: myDream at 0x11dc9e0>, <DOM Text node "
">]
打印输出子节点 <DOM Element: myDream at 0x11dc9e0>
获得父节点 <DOM Element: dream at 0x11dc918>
parent 和 domRoot 对象是相同的 True
```

```
domRoot 元素的第一个子节点： <DOM Text node "
">
domRoot 元素的最后一个子节点： <DOM Text node "
">
获取同一父节点下的前一个子节点： <DOM Text node "
">
获取同一父节点下的后一个子节点： <DOM Text node "
">
获取 Node 对象的所有属性值： [(u'quality', u'myDream')]
>>>
```

前面介绍了通过元素的属性可以获得 Node 对象的一些信息，当然还可以使用 appendChild 方法创建一个子节点，使用 removeChild 方法移除一个子节点，使用 replaceChild 方法将一个已有的节点替换成另外一个节点。

4. Element和Text接口

Element 类也是 Node 类的子类，因此 Node 中支持的属性和方法在 Element 类中同样可以使用，该接口主要用来实现对元素节点的操作。例如，我们可以使用 Element 接口提供的 tagName 属性来获得节点元素的名称，使用该接口提供的 getAttribute 和 setAttribute 方法来对元素节点的属性进行操作。

Text 实现了 Node 接口，但该文本节点不能再有子节点，目的是用来存储数据，在这里不再详细举例。

13.4.3　实例描述

每个人都有好奇心，我也不例外。在.NET 中有这样的情况：将数据库连接字符串存放到一个 XML 文件中，之后使用某个方法调用。在刚接触 Python 时，我就有想要使用 XML 存储数据的冲动，更何况现在学习了如何解析 XML 数据，我更加控制不住自己了，因此我将数据库的连接存储到一个 XML 文件中，接着使用 DOM 中的方法将 XML 文件中的数据解析出来，下面看看我的实现方案。

13.4.4　实例应用

【例 13-3】从 XML 文件中读取数据库配置。

(1) 创建一个名称为 connection.xml 的文件。

(2) 在 connection.xml 文件中添加代码。

```
<DBConnections>
   <DBConnection Key="Test" Server="192.168.0.9" Database="Test"
User="Testsa" Password="testsa"/>
   <DBConnection Key="Ds" Server="192.168.0.6" Database="Test" User="Dsdcy"
Password="dsdcy"/>
```

```
    <DBConnection Key="User" Server="(local)" Database="Master" User="Usersa"
Password="usersa"/>
    <DBConnection Key="Local" Server="192.168.0.5" Database="Test"
User="Localsa" Password="localsa"/>
 </DBConnections>
```

(3) 创建一个名称为 conf.py 的文件，并在该文件中添加代码。

```
import xml.dom.minidom
#从 XML 文件读取数据库配置的类
class CDBConfig:
  #ConfigFilePath 是配置文件的路径
  def __init__(self,ConfigFilePath):
    self.__ConfigFilePath=ConfigFilePath
    #CDBConfig.DBConnects 是 CDBConfig 类的静态字典成员，用来存放数据库访问串
    CDBConfig.DBConnects={}
    #从 XML 文件中读取数据可连接信息，定义往连接数据字典里增加连接的方法
    self.ConfigXMLFile()
  def
AddConnect(self,key,server="(localhost)",database="master",user="sa",passwo
rd="",dbtype="SQLServer"):
    if dbtype=="SQLServer" :
      self.__sconn = "server=" + server + ";database=" + database + ";uid=" +
user +";pwd=" + password;
      CDBConfig.DBConnects[key]=self.__sconn
  #定义读取 XML 文件的方法
  def ConfigXMLFile(self):
    self.__key = ""
    self.__server = ""
    self.__database = ""
    self.__user = ""
    self.__password = ""
    self.__xmlFile=open(self.__ConfigFilePath,'r')       #只读打开配置文件
    self.__dom=xml.dom.minidom.parse(self.__xmlFile)     #解析 xml
    self.__xmlFile.close()                               #关闭文件
    #取得所有的 DBConnection 节
    self.__connect_elements=self.__dom.getElementsByTagName("DBConnection")
    for connect_element in self.__connect_elements:                  #每个
DBConnection 节点都是一个连接串
        self.__key      = connect_element.getAttribute("Key")
        self.__server   = connect_element.getAttribute("Server")
        self.__database = connect_element.getAttribute("Database")
        self.__user     = connect_element.getAttribute("User")
        self.__password = connect_element.getAttribute("Password")
        self.AddConnect(self.__key,self.__server,self.__database,
self.__user,self.__password,dbtype="SQLServer")
        print '---------key 为',self.__key,'的 DBConnection----------'
        print self.__key
        print self.__server
        print self.__database
```

```
        print self.__user
        print self.__password
if __name__=="__main__":
    myconns=CDBConfig("connection.xml")
    for key in CDBConfig.DBConnects.keys():#对每一个 key 执行
      print '-------------打印每个数据库的连接--------------'
      print key+'\t'                                    #把每个数据库连接都打印出来
```

(4) 保存修改好的代码。

13.4.5　运行结果

运行程序，执行结果如下：

```
>>>
---------key 为 Test 的 DBConnection----------
Test
192.168.0.9
Test
Testsa
testsa
---------key 为 Ds 的 DBConnection----------
Ds
192.168.0.6
Test
Dsdcy
dsdcy
---------key 为 User 的 DBConnection----------
User
(local)
Master
Usersa
usersa
---------key 为 Local 的 DBConnection----------
Local
192.168.0.5
Test
Localsa
localsa
-------------打印每个数据库的连接--------------
Test
-------------打印每个数据库的连接--------------
Local
-------------打印每个数据库的连接--------------
Ds
-------------打印每个数据库的连接--------------
User
>>>
```

13.4.6 实例分析



源码解析

在本实例中，通过使用 getElementsByTagName 方法获取配置文件中所有的 DBConnection 节点，接着使用 getAttribute 方法来获得该元素节点的属性值，最后循环遍历该节点下的属性值，这样就将数据库连接打印输出了。

13.5 可扩展样式表语言XSL

提及可扩展样式表语言，是不是有些迷惑？你可能会说：我学过的 XML 技术是一种可扩展标记语言，那么什么是可扩展样式表语言呢？别急，可扩展样式表语言即 XSL(Extensible Stylesheet Language)，它是一种 XML 语汇表，其最主要用途就是将 XML 文档转换成 HTML 格式的文件，然后交付给浏览器，再由浏览器显示转换的结果。下面我们就来看一下 XSL 是如何转换 XML 文档的。



视频教学：光盘/videos/13/ XSL 的使用.avi　　长度：13 分钟

通过上面对 XSL 的介绍，了解了什么是 XSL 以及 XSL 的用处。接下来看一个小例子。

首先，创建一个名称为 myXML.xml 的文件，并添加如下代码：

```
<?xml version="1.0"?>
<!-- XMLdocument containing book information. -->
<?xml-stylesheet type="text/xsl" href="my.xsl"?>
<book isbn="123654">
    <title>亲爱的 XML 文档</title>
    <author>
        <firstName> duanchunyang</firstName>
        <lastName> maxianglin</lastName>
    </author>
    <chapters>
        <frontMatter>
            <preface pages="2"/>
            <contents pages="5"/>
            <illustrations pages="4"/>
        </frontMatter>
        <chapter number="2" pages="35">
            Intermediate XML</chapter>
        <appendix number="A" pages="7">
            Entities</appendix>
        <chapter number="1" pages="28">
            XML Fundamentals</chapter>
    </chapters>
```

```
</book>
```

接着创建一个名称为 my.xsl 文件，并为该 xsl 文件添加如下模板代码：

```
<div style="width:350px">
     <h2 style="color:#F60" align="center"><xsl:value-of select =
"title"/></h2>
     <table style = "border-style: solid; background-color: wheat "
border="1px" width="350px" align="left">
       <tr>
         <td align="center"><xsl:value-of select = "author/lastName" /></td>
         <td align="center"><xsl:value-of select = "author/firstName" /></td>
       </tr>
       <xsl:for-each select = "chapters/frontMatter/*">
         <tr>
           <td style = "text-align: right"><xsl:value-of select = "name()" /></td>
           <td> ( <xsl:value-of select = "@pages" /> pages ) </td>
         </tr>
       </xsl:for-each>
       <xsl:for-each select = "chapters/chapter">
         <xsl:sort select = "@number" data-type = "number" order= "ascending" />
         <tr>
           <td style = "text-align: right"> Chapter <xsl:value-of select =
"@number" /></td>
           <td> ( <xsl:value-of select = "@pages" /> pages ) </td>
         </tr>
       </xsl:for-each>
       <xsl:for-each select = "chapters/appendix">
         <xsl:sort select = "number" data-type = "text" order = "ascending" />
         <tr>
           <td style = "text-align: right"> Appendix <xsl:value-of select =
"@number" /></td>
           <td> ( <xsl:value-of select = "@pages" /> pages ) </td>
         </tr>
       </xsl:for-each>
     </table>
    </div>
```

运行 myXML.xml 文件，执行效果如图 13-6 所示。

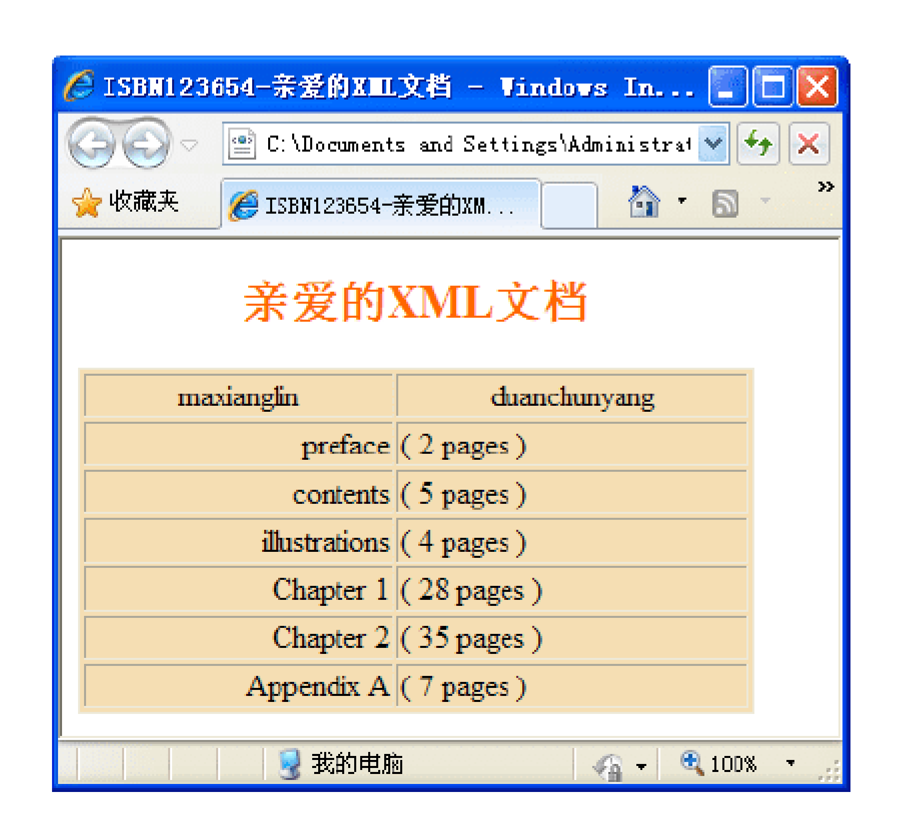


图 13-6　运行XML文件的显示效果

13.6　动态定义树状结构图

你对 JavaScript 熟悉吗？如果熟悉，你肯定不会对 Eval 函数感到陌生，经常使用该函数来实现动态代码的执行。这里要介绍的是如何使用前面学到的知识动态生成 XML 的内容，下面通过一个实例来说明。


视频教学：光盘/videos/13/动态定义树状结构图.avi　　长度：3 分钟

13.6.1　实例描述

古人说：一年之计在于春。春天寄予我们最美好的希望，因此也就有了很多描写春天美好的诗句。在这里我们将这些诗句以 XML 文档结构的形式展现出来，以“美丽诗句”为根节点，以“春天”为子节点，并且在子节点中再次添加子节点，以此来说明春天带来的特点。下面就是我的实现方案。

13.6.2　实例应用

【例 13-4】动态定义树状结构图。

(1) 创建一个名称为 spring.py 的文件。

(2) 在 spring.py 文件中添加代码。

```
from xml.dom.minidom import Document
# 声明一个 DOM 结构
doc = Document()
# 创建一个根元素“美丽诗句”
wml = doc.createElement("美丽诗句")
doc.appendChild(wml)
# 在根元素“美丽诗句”下添加子元素“春天”
maincard = doc.createElement("春天")
maincard.setAttribute("id", "main")
```

```
wml.appendChild(maincard)
#在子元素“春天”下添加子节点“咏柳”
paragraph1 = doc.createElement("咏柳")
maincard.appendChild(paragraph1)
#在子节点“咏柳”下添加说明
ptext = doc.createTextNode("碧玉妆成一树高，万条垂下绿丝绦")
paragraph1.appendChild(ptext)
#在子元素"春天"下添加子节点“游园不值”
paragraph1 = doc.createElement("游园不值")
maincard.appendChild(paragraph1)
#在子节点“游园不值”下添加说明
ptext = doc.createTextNode("春色满园关不住，一枝红杏出墙来")
paragraph1.appendChild(ptext)
#在子元素“春天”下添加子节点“春日”
paragraph1 = doc.createElement("春日")
maincard.appendChild(paragraph1)
#在子节点“春日”下添加说明
ptext = doc.createTextNode("等闲识得东风面，万紫千红总是春")
paragraph1.appendChild(ptext)
# 打印输出 XML 文件中的内容
print doc.toprettyxml(indent=" ")
```

(3) 保存修改好的代码。

13.6.3　运行结果

运行程序，执行结果如下：

```
>>>
<?xml version="1.0" ?>
<美丽诗句>
 <春天 id="main">
  <咏柳>
   碧玉妆成一树高，万条垂下绿丝绦
  </咏柳>
  <游园不值>
   春色满园关不住，一枝红杏出墙来
  </游园不值>
  <春日>
   等闲识得东风面，万紫千红总是春
  </春日>
 </春天>
</美丽诗句>

>>>
```

13.6.4 实例分析


源码解析

在本实例中，创建了 Document 接口的实例对象 doc，接着使用 createElement 方法创建了一个根节点，然后使用 appendChild 方法将该根节点添加到 XML 文档中，最后使用 doc 的 toprettyxml 方法将文档的内容打印输出。

13.7 常见问题解答

13.7.1 SAX解析XML问题



SAX 解析 XML 问题。

网络课堂：http://bbs.itzcn.com/thread-15638-1-1.html

我有这样一段代码：

```
from xml.sax import*
class XmlHandler(ContentHandler):
    def startDocument(self):
        print '--- Begin Document ---'
    def startElement(self, name, attrs):
        if name == 'Msg':
            print 'begin Msg',
    def endElement(self, name):
        if name == 'Msg':
            print 'end',name
    def characters(self, ch):
        print ch,
if __name__ == '__main__':
    handler = XmlHandler()
    xmlStr = '''<?xml version="1.0" encoding="GB2312" standalone="yes" ?>
<Msg Version="1" MsgID="2">
<Query Inst="0" Date="2006-05-12" />
</Msg>'''
    parseString(xmlStr,handler)
```

执行结果显示如下错误：

```
>>>
--- Begin Document ---

Traceback (most recent call last):
  File "F:/Python/第 13 章/node.py", line 20, in <module>
    parseString(xmlStr,handler)
  File "D:\Program Files\Python\lib\xml\sax\__init__.py", line 49, in
parseString
    parser.parse(inpsrc)
```

```
  File "D:\Program Files\Python\lib\xml\sax\expatreader.py", line 107, in parse
    xmlreader.IncrementalParser.parse(self, source)
  File "D:\Program Files\Python\lib\xml\sax\xmlreader.py", line 123, in parse
    self.feed(buffer)
  File "D:\Program Files\Python\lib\xml\sax\expatreader.py", line 211, in feed
    self._err_handler.fatalError(exc)
  File "D:\Program Files\Python\lib\xml\sax\handler.py", line 38, in fatalError
    raise exception
SAXParseException: <unknown>:1:30: unknown encoding
>>>
```

但是只要将<?xml version="1.0" encoding="GB2312" standalone="yes" ?>删除就可以解析正确，如果说这句话不能写入程序，但在 XML 文档中却可以出现，该如何解释呢？请各位指点一二。

【解决办法】很高兴为你解答。

出现上面的错误是你的编码问题。在调用 parseString 方法前，需要将 GB2312 编码转换为 UTF-8，因为 SAX 库并不支持 GB2312 编码。修改之后执行的结果如下：

```
>>>
--- Begin Document ---
begin Msg

end Msg
>>>
```

怎么样？清楚了吧。

13.7.2　DOM中的xml.dom.minidom问题



DOM 中的 xml.dom.minidom 问题。

网络课堂：http://bbs.itzcn.com/thread-15639-1-1.html

我有个 XML 文件，大概是这样的：

```
<TEXT>
<s num = '1' count = '23' >for currentLine in file.readlines</s>
<s num = '2' count = '23' >for currentLine in file.readlines</s>
<s num = '3' count = '23' >for currentLine in file.readlines</s>
<s num = '4' count = '23' >for currentLine in file.readlines</s>
<s num = '5' count = '23' >for currentLine in file.readlines</s>
</TEXT>
```

我想把<TEXT>中的<s>的属性 num 和 count 全部取出来，但因每个文件的<s>属性数目不定，不知道该怎么办才好？希望大家不吝赐教，谢谢！

【解决办法】很高兴为你解答。

根据你的要求，我写了一段代码，希望是你想要的，代码如下：

```
from xml.dom import minidom
xml_string = '''
<TEXT>
<s num = '1' count = '123' name='dcy'>for currentLine in file.readlines </s>
<s num = '2' count = '23' >for currentLine in file.readlines </s>
<s num = '3' count = '23' name='mxl'>for currentLine in file.readlines </s>
```

```
<s num = '4' count = '23' >for currentLine in file.readlines </s>
<s num = '5' count = '23' >for currentLine in file.readlines </s>
</TEXT>
'''
doc = minidom.parseString(xml_string)
nodes = doc.getElementsByTagName("s")
for n in nodes:
    print '在 s 标签中第一个属性是: ',n.getAttribute('num'), '在 s 标签中第二个属性是:
',n.getAttribute('count'),'在 s 标签中第三个属性是: ',n.getAttribute('name')
```

运行程序，执行结果如下：

```
>>>
在 s 标签中第一个属性是:  1 在 s 标签中第二个属性是:  123 在 s 标签中第三个属性是:  dcy
在 s 标签中第一个属性是:  2 在 s 标签中第二个属性是:  23 在 s 标签中第三个属性是:
在 s 标签中第一个属性是:  3 在 s 标签中第二个属性是:  23 在 s 标签中第三个属性是:  mxl
在 s 标签中第一个属性是:  4 在 s 标签中第二个属性是:  23 在 s 标签中第三个属性是:
在 s 标签中第一个属性是:  5 在 s 标签中第二个属性是:  23 在 s 标签中第三个属性是:
>>>
```

瞧，上面这样的效果是你想要的吧。

13.7.3　动态生成XML文档问题



动态生成 XML 文档问题。
网络课堂：http://bbs.itzcn.com/thread-15640-1-1.html

有这样一段代码：

```
from xml.dom.minidom import Document
xmltree = Document()
firstNode = xmltree.createElement("newslist")
xmltree.appendChild(firstNode)
d="我们彼此依偎，等我，加油！"
n=xmltree.createTextNode(d)
firstNode.appendChild(n)
outf = open("test.xml","w")
outf.write(xmltree.toprettyxml(indent="", newl="", encoding="UTF-8"))
outf.close()
```

上面这段代码是动态生成 XML 文件，并且将中文输入到 XML 文件中。可能是由于中文的关系，程序报错。希望各位能给出改正后的代码。

【解决办法】很高兴为你解答。

以下是我给你的代码：

```
#coding=gb18030
import sys
reload(sys)
sys.setdefaultencoding('gb18030')
from xml.dom.minidom import Document

xmltree = Document()
firstNode = xmltree.createElement("newslist")
xmltree.appendChild(firstNode)
```

```
d="我们彼此依偎，等我，加油！"
n=xmltree.createTextNode(d)
firstNode.appendChild(n)

outf = open("test.xml","w")
outf.write(xmltree.toprettyxml(indent="", newl="", encoding="UTF-8"))
outf.close()
```

执行程序，并且在该目录下打开 test.xml 文件，显示效果如图 13-7 所示。

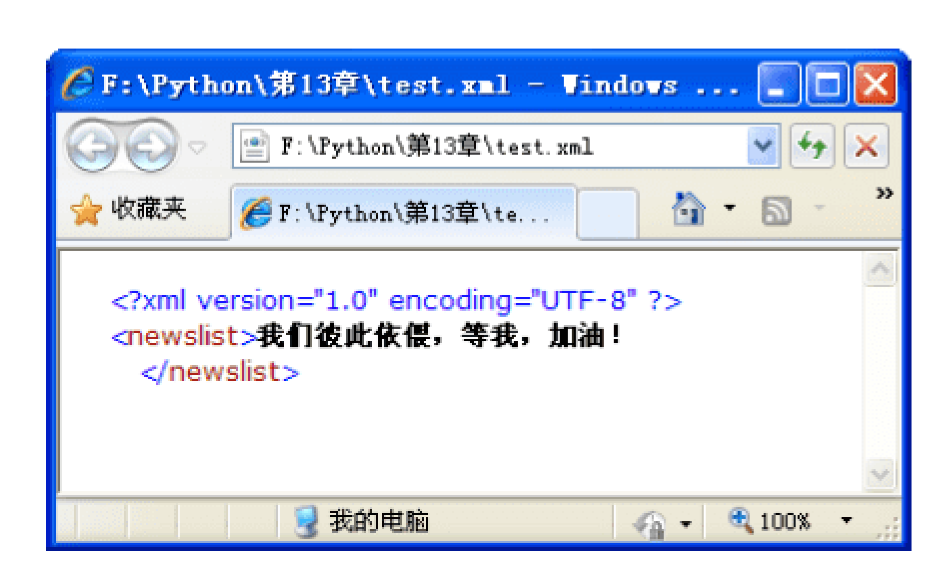


图 13-7 生成XML文档的效果

13.8 习 题

一、填空题

(1) 关于 XML 中的字符和实体引用，其中<字符在文档中显示的效果是<，那么字符&gt在文档中显示的效果是__________。

(2) __________是用来包含文本的方法，通常用于建立代码的脚本有 JavaScript、VBScript 等。

(3) ContentHandler 接口是在使用 SAX 技术时用得最多的处理器对象，当处理 XML 文档时，首先调用的方法是__________。

(4) 在使用 SAX 解析器解析 XML 数据时遇到的每个元素，__________方法所注册的函数都会被调用。

(5) 在解析 XML 时，当遇到需要处理有关 DTD 定义的内容，我们可以使用__________接口的实例对象。

(6) 在 Python 标准库中支持 DOM 操作的模块是 xml.dom，如果应用程序想要判断 DOM 实现是否支持该特性，那么 xml.dom 模块需要实现的接口是__________。

(7) 有这样一段代码：

```
from xml.dom.minidom import *
implementation=DOMImplementation()
documentype=__________.createDocumentType('happy','','happy')
document=implementation.createDocument('','happies',documentype)
```

该段代码的主要目的是生成 Document 的实例对象，那么在空白处需要填写的对象是__________。

(8) Node 指的是文档结构的每个单节点。在 DOM 中的结构都是 Node 对象，每个节点的类型通过__________来区别。

二、选择题

(1) 在使用 SAX 解析器解析 XML 文档数据时，__________接口的处理器对象是最常用的。

A. ContentHandler　　B. DTDHandler

C. XMLReader　　D. EntityResolver

(2) 在 xml.dom 模块实现的接口中，下面选项中__________是表示整个 XML 文档的。

A. DOMlmplemetation　　B. Document

C. Node　　D. Element

(3) 有这样一段代码:

```
from xml.dom.minidom import *
dom=parse('my.xml')
domRoot=dom.documentElement
childs=domRoot.childNodes
mydream=domRoot.childNodes[1]
parent=mydream.parentNode
attr=mydream.attributes
print parent
print parent==domRoot
print domRoot.firstChild
print domRoot.lastChild
print attr.items()
```

其中 XML 文件中的代码如下:

```
<dream>
<myDream quality='myDream'>
<say>you are the best one</say>
</myDream>
</dream>
```

运行程序，下面几个选项中表示执行的结果正确的是__________。

A.

```
>>>
<DOM Element: dream at 0x11dc878>
False
<DOM Text node "
">
<DOM Text node "
">
[(u'quality', u'myDream')]
>>>
```

B.

```
>>>
<DOM Element: dream at 0x11dc878>
True
<DOM Text node "
">
<DOM Text node "
">
[]
>>>
```

C.

```
>>>
<DOM Element: dream at 0x11dc878>
True
<DOM Text node "
">
<DOM Text node "
">
[(u'quality', u'myDream')]
>>>
```

D.

```
>>>
<DOM Element: dream at 0x11dc878>
True
<DOM Text node "
">
None
[(u'quality', u'myDream')]
>>>
```

(4) 下面几个选项中的代码是使用 DOMImplemetation 接口对象的某些方法生成 Document 接口对象的，正确的是__________。

A.

```
from xml.dom.minidom import DOMImplementation
documentype= DOMImplementation.createDocumentType('HI','','HI')
document= DOMImplementation.createDocument('','HI',documentype)
print document
```

B.

```
from xml.dom.minidom import DOMImplementation
implementation=DOMImplementation()
```

```
documentype=Document.createDocumentType('HI','','HI')
document= Document.createDocument('','HI',documentype)
print document
```

C.

```
from xml.dom.minidom import DOMImplementation
implementation=DOMImplementation()
documentype=implementation.createDocumentType('HI','','HI')
document=implementation.createDocument('','HI')
print document
```

D.

```
from xml.dom.minidom import DOMImplementation
implementation=DOMImplementation()
documentype=implementation.createDocumentType('HI','','HI')
document=implementation.createDocument('','HI',documentype)
print document
```

(5) 在动态生成 XML 文档时，我们通过__________方法来创建一个子节点，下面的选项正确的是__________。

A. removeChild　　B. appendChild　　C. replaceChild　　D. childNode

三、上机练习

上机练习： 对 XML 文档增、删、改、查操作。

通过对本章的学习，我们不仅对 XML 文档结构有了大致的了解，还对 Python 与 XML 交互有了进一步的认识。例如，使用 xml.sax 模块中的接口对 XML 数据进行处理，通过 DOM 将 XML 数据解析成树状结构，通过 DOM 中的 Document 接口动态定义 XML 文档结构等。本次上机练习就是针对 XML 文档操作。

有这样一个 XML 文档，代码如下：

```
<?xml version="1.0" encoding="utf-8"?>
<Player>
<Name>DCY</Name>
<Age>23</Age>
<Shu>mouse</Shu>
<HighSchool>SUIXIAN</HighSchool>
<Phone>15093077823</Phone>
<Hoppy>lvyou</Hoppy>
</Player>
```

首先需要将该 XML 文档中的内容全部读取出来，之后要求对 XML 文档中的节点和内容进行增、删、改、查等操作。



第 14 章　图形用户界面

内容摘要

wxPython 是 Python 语言的一个 GUI(Graphical User Interface，图形用户界面)工具集，它可以使 Python 程序员简单而轻松地创建具有高级功能的图像用户界面。它在 Python 中是以扩展模块的方式实现的，拥有自身的窗体/控制，还加入了许多独立于操作系统的窗口，并且封装了流行的 Windows 跨平台 GUI 库。wxPython 同样具有跨平台能力，这标志着编写出来的代码可以不经修改地运行在绝大多数操作系统之上，提高了代码的利用率。

本章将重点介绍 wxPython 的开发环境、基本组件和高级控件。

学习目标

- 熟悉 wxPython 的程序结构。
- 掌握 wxPython 的基本组件和常用组件。
- 掌握 wxPython 库中的菜单、窗口与对话框控件。
- 掌握 wxPython 库中的表格控件。
- 掌握 wxPython 库中的高级列表控件。
- 掌握 wxPython 库中的树型控件。
- 掌握 wxPython 库中的定时器控件。

14.1 wxPython的开发环境

wxPython 是跨平台 GUI 图形界面开发库 wxWidgets 在 Python 中的语言绑定。其中含有丰富的窗体部件类，使得用户可以快速地创建功能强大、界面元素丰富的 GUI 应用程序。同时，wxPython 具有跨平台特性，也就是说，同一个应用程序不经过修改就可以运行在多种系统平台下，包括 Windows、Linux 和 Macintosh 等。wxPython 为使用此开发库的应用程序提供了基于相应系统平台的外观。在 Windows 下，会使用 Windows 的窗体效果；在 Linux 下，会使用桌面环境下的窗体效果。本节将介绍 wxPython 的下载和安装步骤。

视频教学：光盘/videos/14/wxPython 的开发环境.avi　　长度：8 分钟

14.1.1 基础知识——丰富的平台

使用 Python 语言，需要决定使用哪种 GUI 平台。简单来说，平台是图形组件的一个特定集合，可以通过叫做 GUI 工具包的给定 Python 模块进行访问。Python 可用的工具包很多，其中一些最流行的工具包如表 14-1 所示。

表 14-1　支持Python的流行GUI工具包

工 具 包	描 述
Tkinter	Tkinter 似乎是与 TCL(Tool Command Language，工具命令语言)语言同时发展起来的一种界面库。Tkinter 是为 Python 配备的标准 GUI 库，也是 opensource 的产物。Tkinter 可用于 Windows、LINUX、UNIX 和 Macintosh 操作系统，而且显示风格是本地化的。Tkinter 用起来非常简单，Python 自带的 IDLE 就是采用它写的。此外，Tkinter 的扩展集 pmw 和 Tix 功能上都比它强大，但 Tkinter 却是最基本的
wxWidget	wxWidgets 算是近几年比较流行的 GUI 跨平台开发技术。wxWidgets 有不同的版本，有 C++ 的，也有 Basic 的，在 Python 上也有较好的移植。wxWidgets 的功能要强于 Tkinter，它提供了超过 200 个类，面向对象的编程风格，设计框架类似于 MFC。对于大型 GUI 应用，wxWidgets 具有很强的优势。boa constructor 可以帮助我们快速可视地构建 wxWidgets 界面
PythonWin	只能在 Windows 上使用，使用了本机的 Windows GUI 功能
Java Swing	只能用于 Jython，使用本机的 Java GUI
PyGTK	PyGTK 是 Linux 下 Gnome 的核心开发库，功能非常齐全。值得说明的是，在 Windows 平台下 PyGTK 的显示风格并不是特别本地化。PyGTK 使用 GTK 平台，在 Linux 上很流行
PyQt	PyQt 同样是一种开源的 GUI 库，PyQt 的类库大约有 300 个，函数大约有 5700 个。PyQt 同样适合于大型应用，它自带的 qt designer 可以让我们轻松构建界面元素

可选的工具包太多，那么到底使用哪个呢？尽管每个工具包都有利有弊，但很大程度上取决于个人喜好。Tkinter 类似于标准 GUI，因为它被用于大多数“正式的”Python GUI 程序，而且它是 Windows 二进制发布版的一部分，在 UNIX 上要自己编译安装。

另外一个越来越受欢迎的工具是 wxPython，这是一个成熟而且特性丰富的包，也是 Python 之父 Guido van Rossum 的最爱。

14.1.2　基础知识——下载和安装wxPython

在使用 wxPython 之前需要为 Python 安装 wxPython 包，步骤如下。

(1) 访问 http://wxpython.org/download.php 页面，下载与 Python 同版本的 wxPython。本书所使用的 Python 版本为 2.5，因此将 wxPython2.8-win32-unicode-2.8.11.0-py25.exe 下载下来即可。

(2) 双击下载好的 wxPython2.8-win32-unicode-2.8.11.0-py25.exe 安装文件，出现图 14-1 所示的欢迎安装界面。

(3) 单击 Next 按钮，进入准备安装界面。在准备安装界面中，提示用户是否同意安装协议，选择 I accept the agreement 单选按钮，如图 14-2 所示。

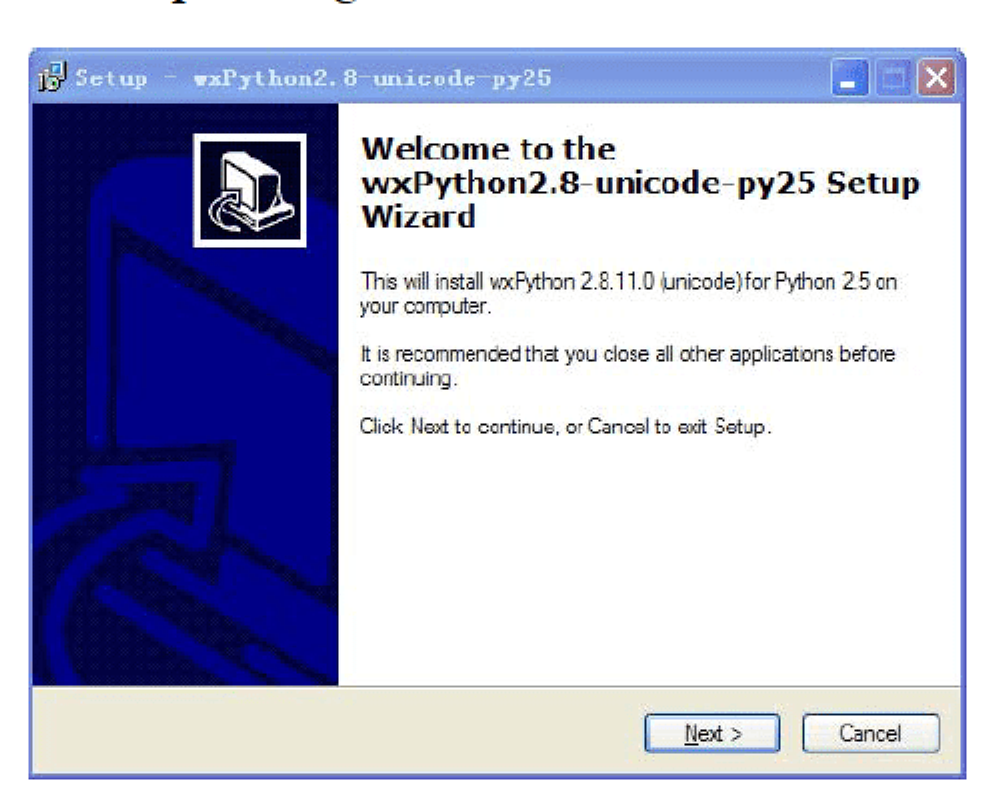


图 14-1　wxPython安装界面

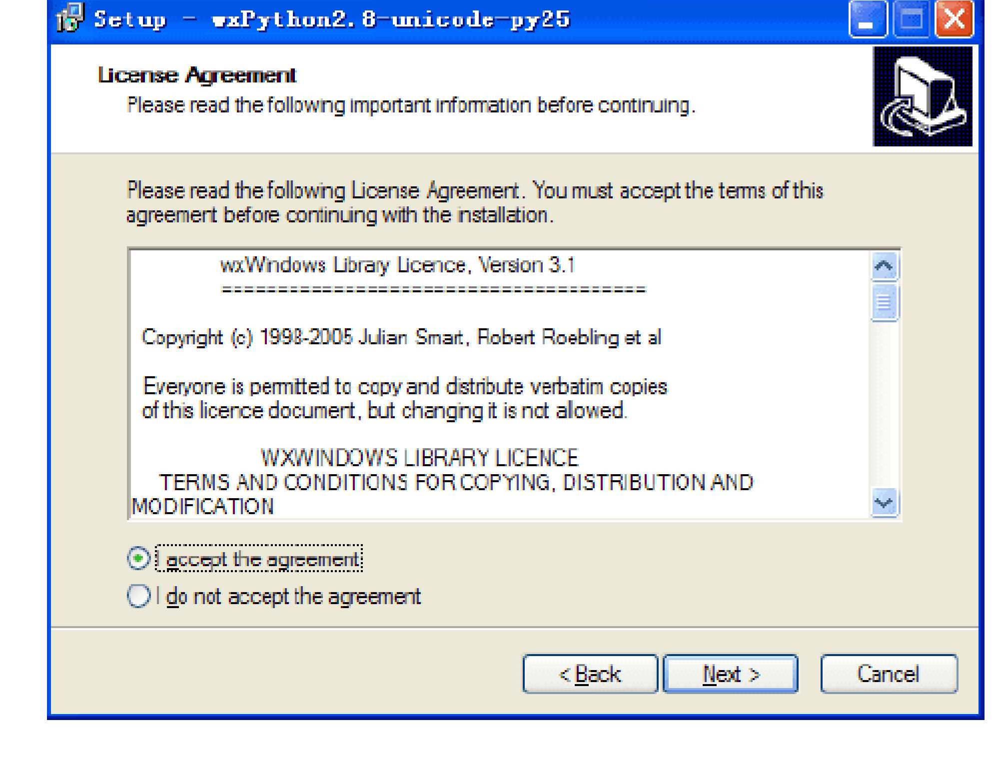


图 14-2　准备安装

(4) 单击准备安装界面中的 Next 按钮之后，进入选择安装路径界面。这里的安装路径为 Python 安装路径下的 Lib/site-packages 目录，将 wxPython 工具包安装于 Python 下，如图 14-3 所示。

(5) 当选择好安装路径之后，单击 Next 按钮，进入选择安装组件的界面。在这里将所有的 wxPython 组件安装，如图 14-4 所示。

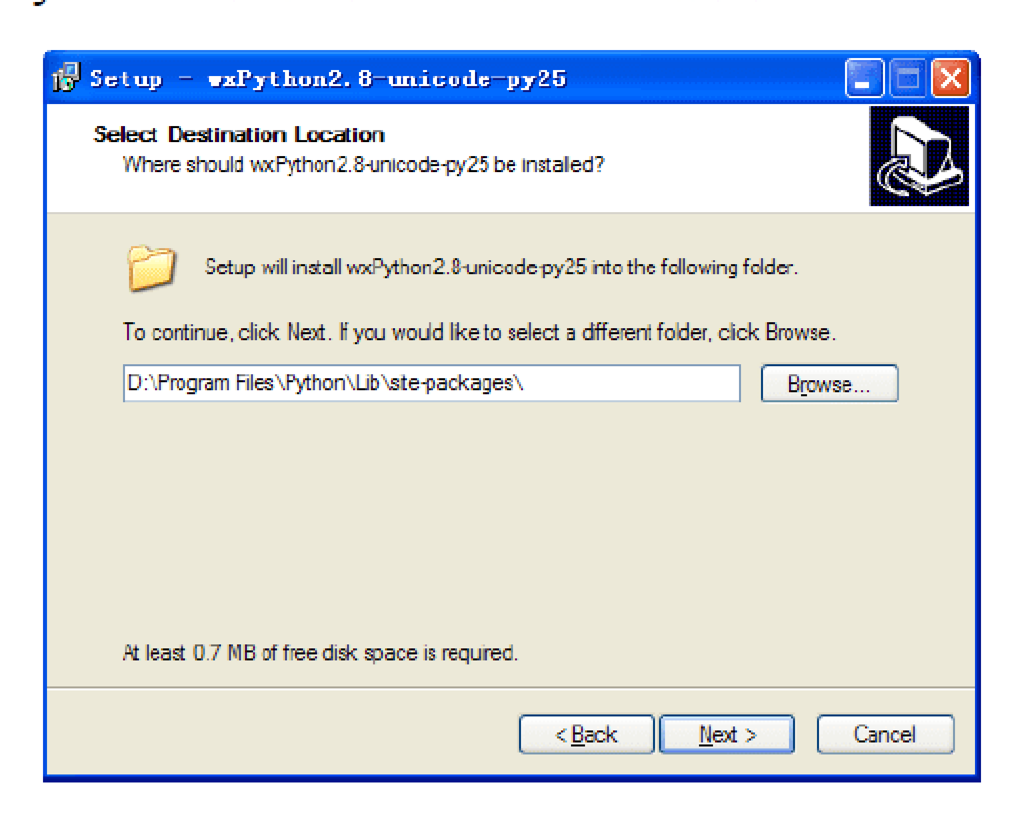


图 14-3　选择安装路径界面

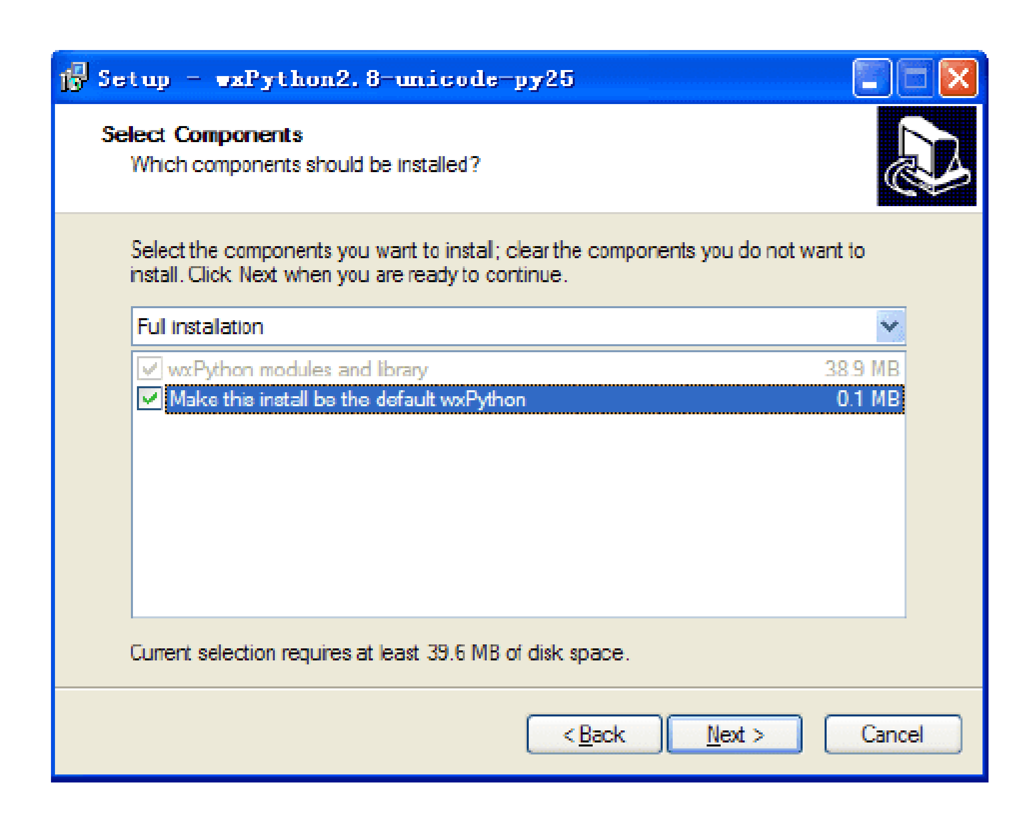


图 14-4　选择安装组件界面

(6) 单击 Next 按钮，进入安装界面，如图 14-5 所示。安装完成后，安装程序自动跳转至完成界面，如图 14-6 所示。

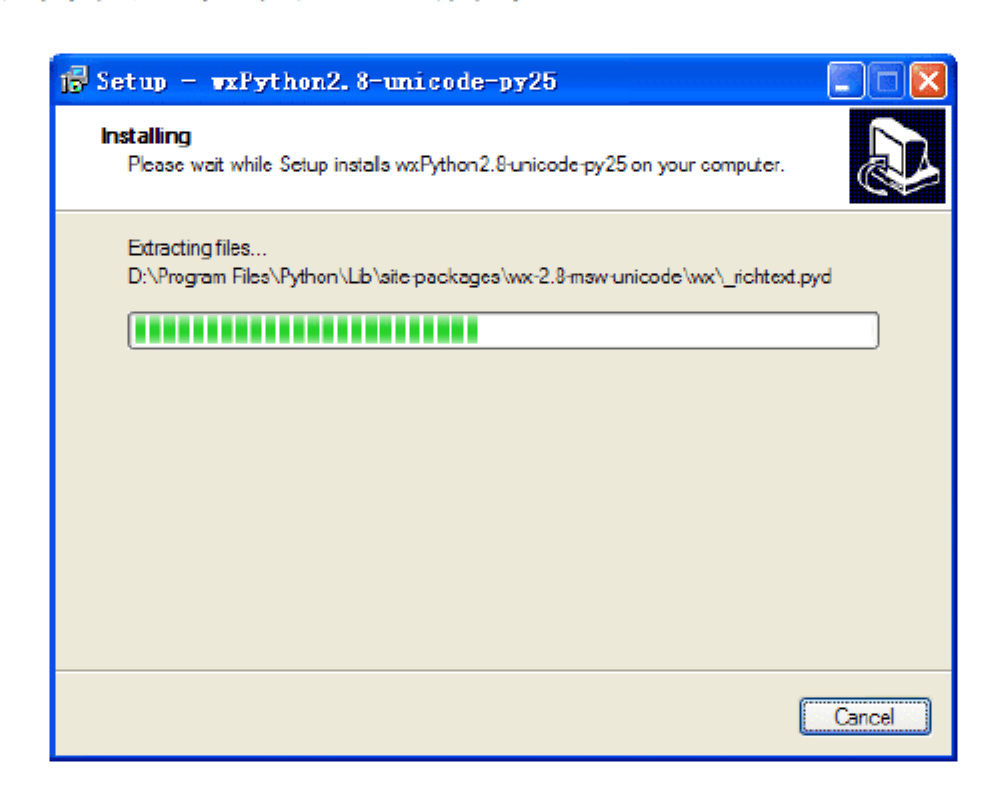


图 14-5　安装界面

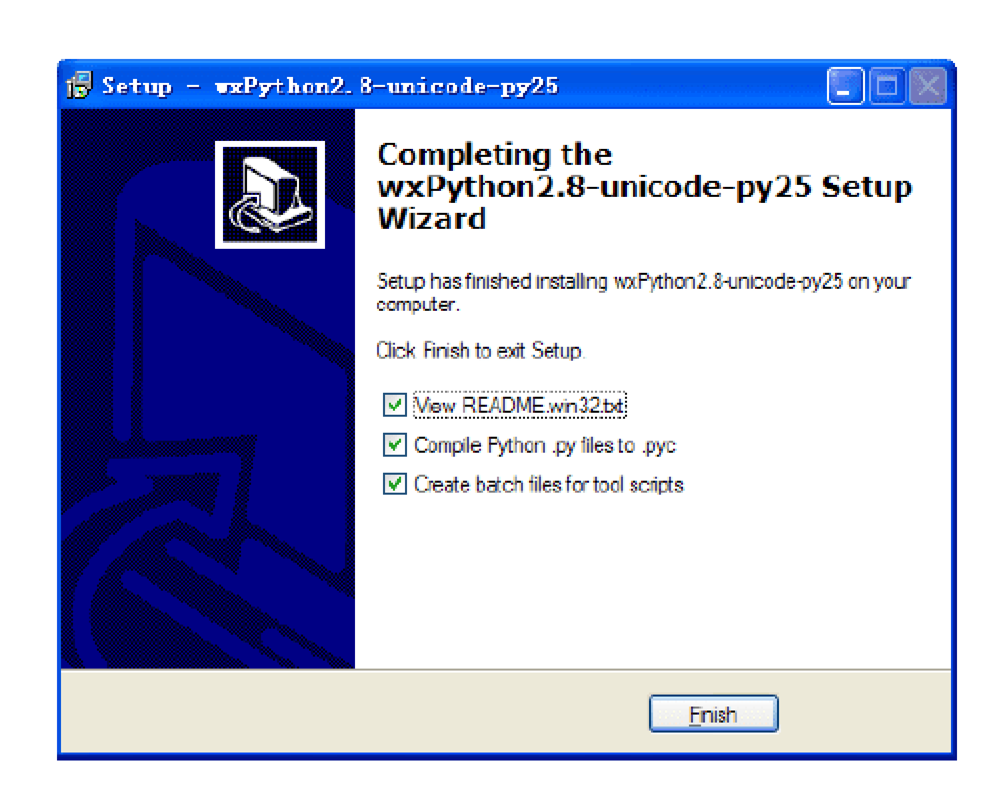


图 14-6　安装完成界面

(7) 单击 Finish 按钮，完成安装。安装完毕后，可以编辑 Python 代码来测试 wxPython 的安装是否成功。

```
import wx
print wx.version()
```

在第一行导入了 wx，所有的 wxPython 下的操作符号都定义在其中。如果此处报错，找不到 wx 模块，则表示 wxPython 的安装没有成功，需要重新安装。在第二行使用了 version()方法来查看当前安装的 wxPython 的版本。运行该段代码，输出结果如下：

```
2.8.11.0 (msw-unicode)
```

14.1.3　基础知识——wxPython的开发工具

wxPython 的开发工具很多，较常用的有 3 个，分别为 WxGlade、WxFormBuilder 和 Boa-constructor。

1．wxGlade开发工具

wxGlade 是一个 GUI 设计工具，其模型来源于知名的 GTK+/GNOME 界面设计工具 Glade，所以 wxGlade 的界面显示和 Glade 十分相似，均采用了多窗口的显示方式。使用 wxGlade 可以生成界面的 Python 代码。

从网上下载 wxGlade 的安装文件 wxGlade-0.6.3-setup.exe，双击该文件进入 wxGlade 的安装程序，如图 14-7 所示。

wxGlade 采用多种窗口的布局形式，包括程序窗口、控件窗口、属性窗口和窗体可视化窗口等。可以直接单击窗口部件来设计 GUI，如图 14-8 所示。

2．wxFormBuilder开发工具

wxFormBuilder 也是一个 GUI 设计工具，从网上下载 wxFormBuilder 的安装文件 wxFormBuilder_v3.0.57.exe，双击安装程序，打开安装界面，如图 14-9 所示。安装完成后，弹

出 wxFormBuilder 的信息页面，如图 14-10 所示。

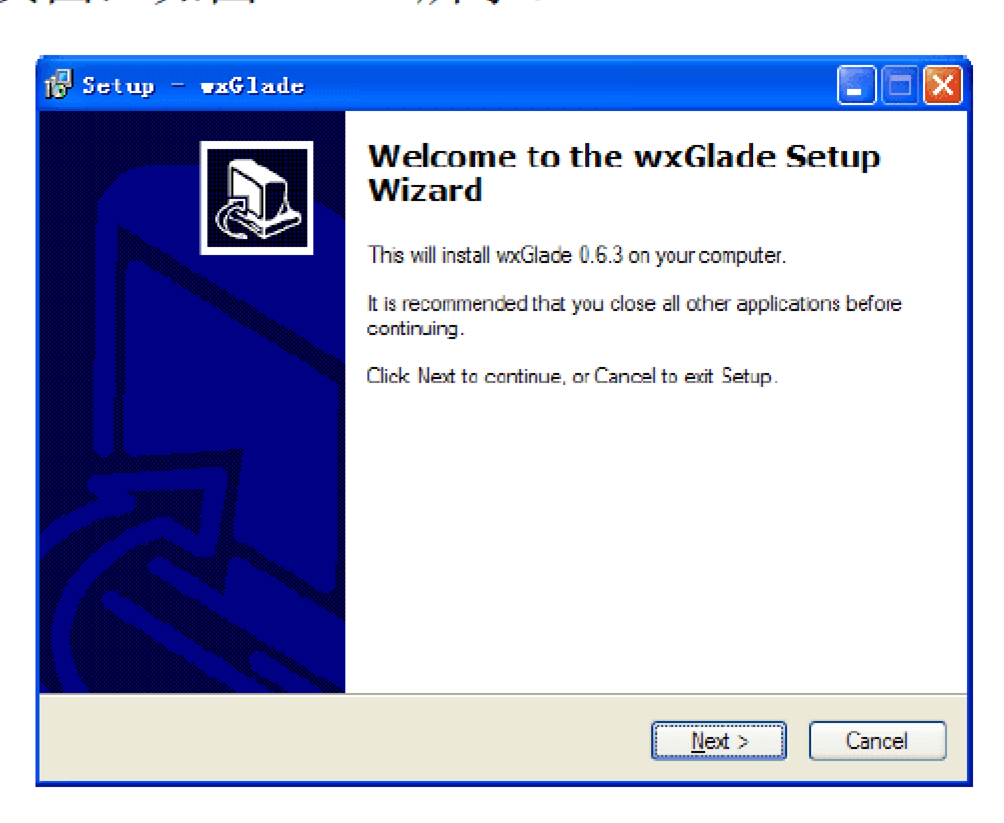


图 14-7　wxGlade的安装欢迎界面

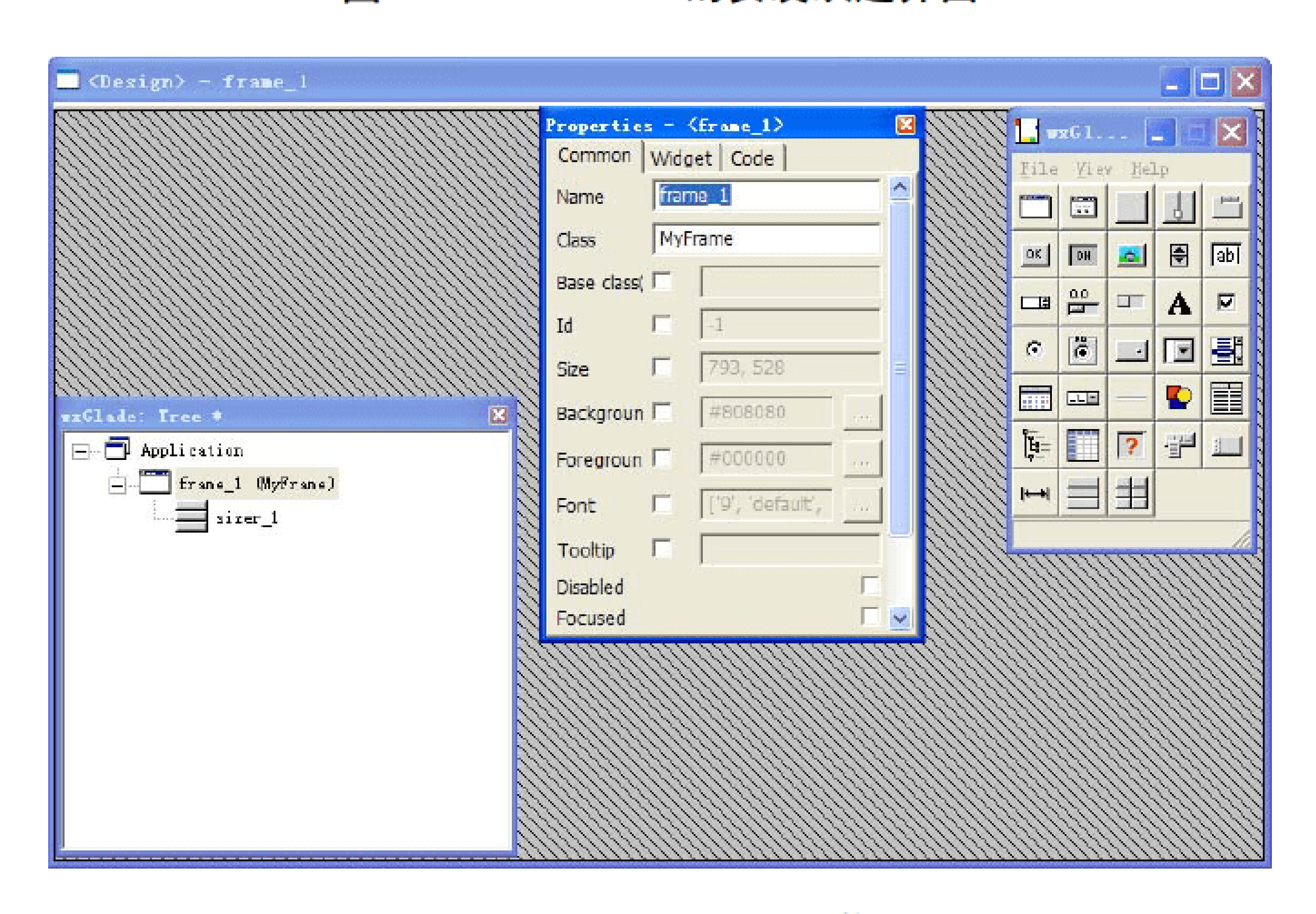


图 14-8　wxGlade程序截图

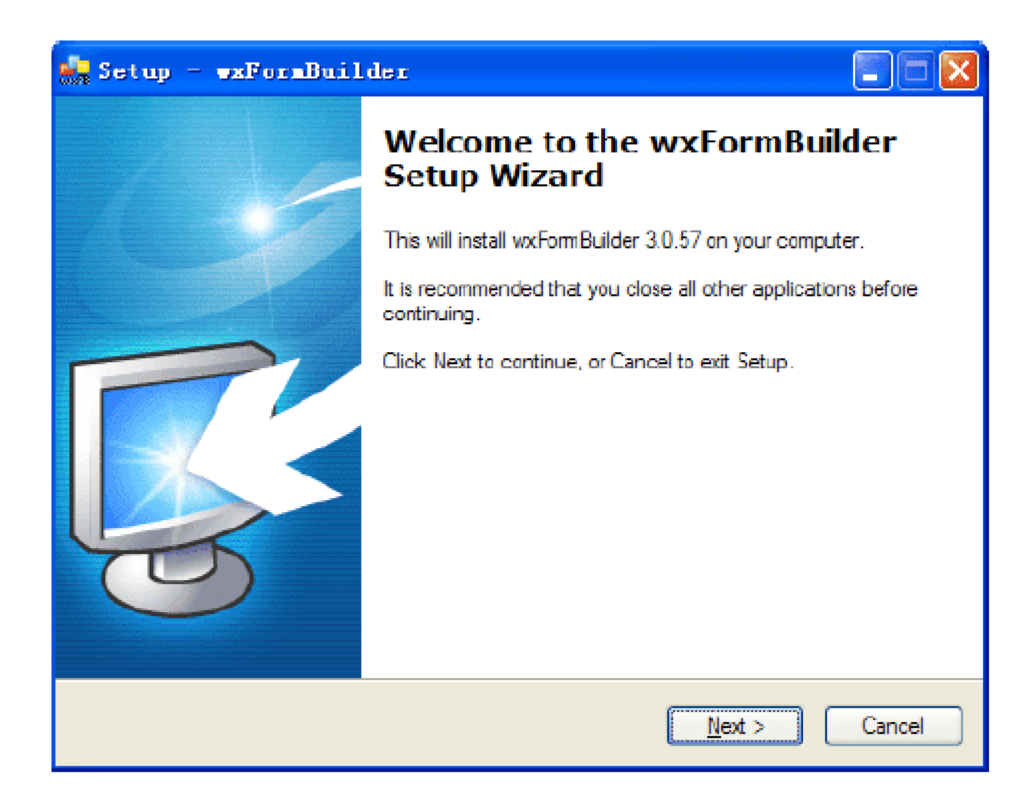


图 14-9　wxFormBuilder的安装欢迎界面

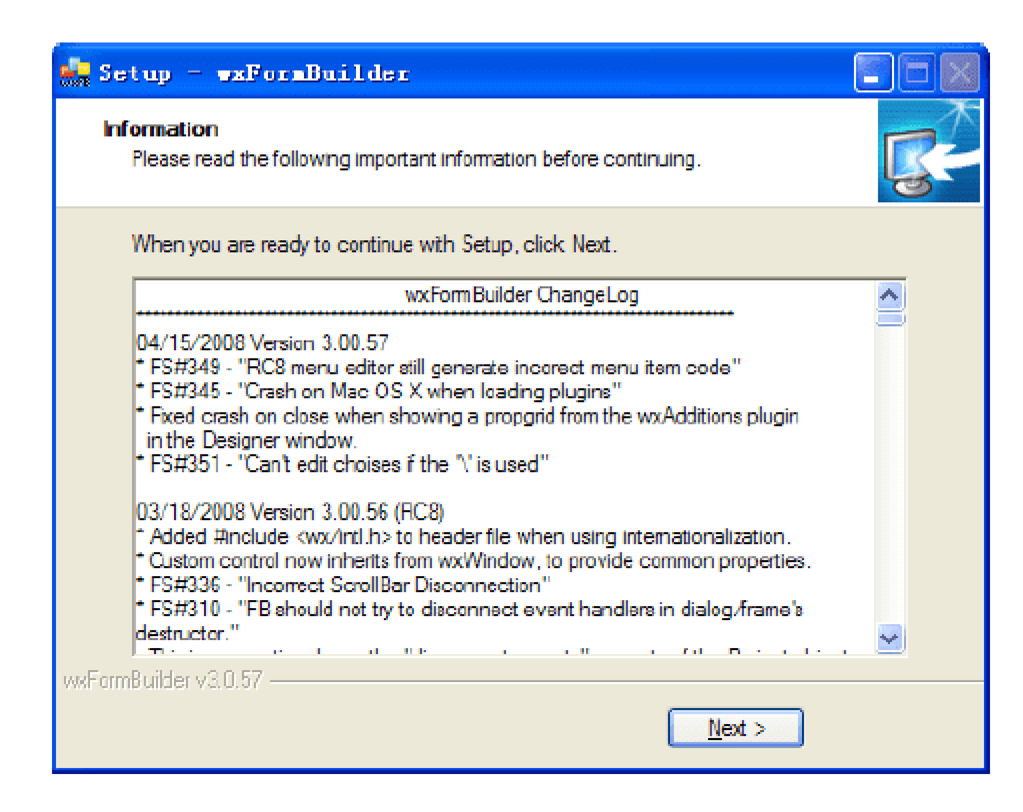


图 14-10　安装完成显示信息界面

安装完毕，打开 wxFormBuilder 程序，如图 14-11 所示。

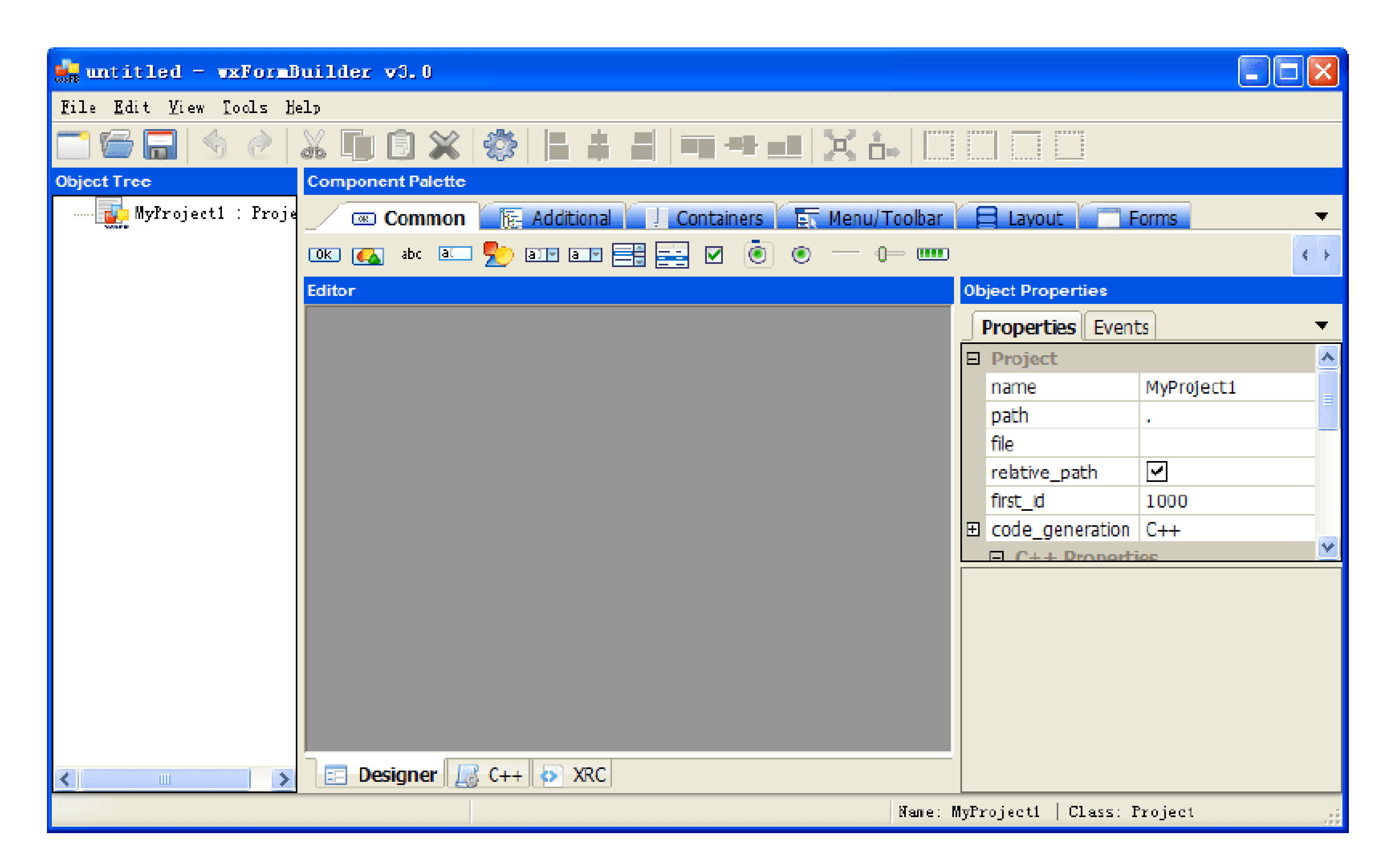


图 14-11　wxFormBuilder程序界面

3．Boa-constructor开发工具

Boa-constructor 是一个 wxPython 的界面设计工具，也是一个强大的 Python 语言 IDE。从网上下载 Boa-constructor 的安装文件 boa-constructor-0.6.1.bin.setup.exe，双击该文件，进入安装界面，如图 14-12 所示。

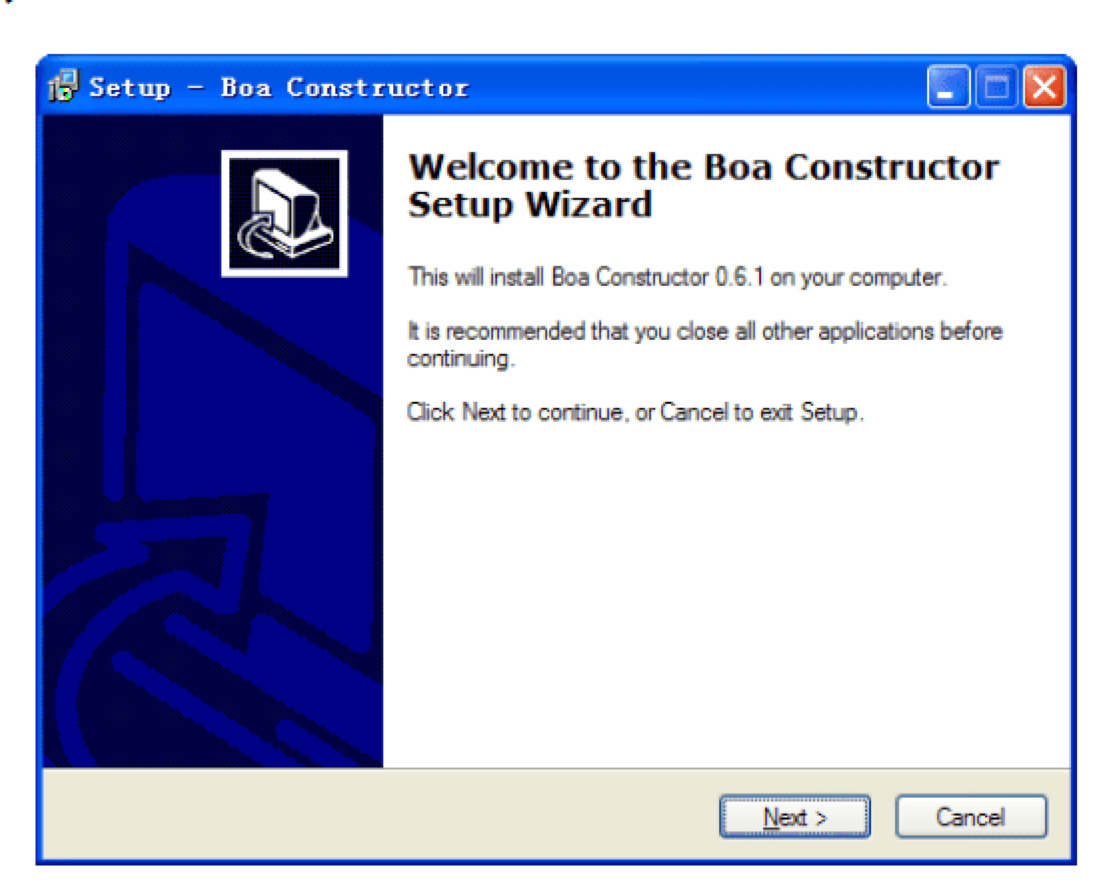


图 14-12　Boa-constructor的安装截图

使用该工具设计 GUI，可以通过单击空间类进行界面设计。Boa-constructor 程序截图如图 14-13 所示。

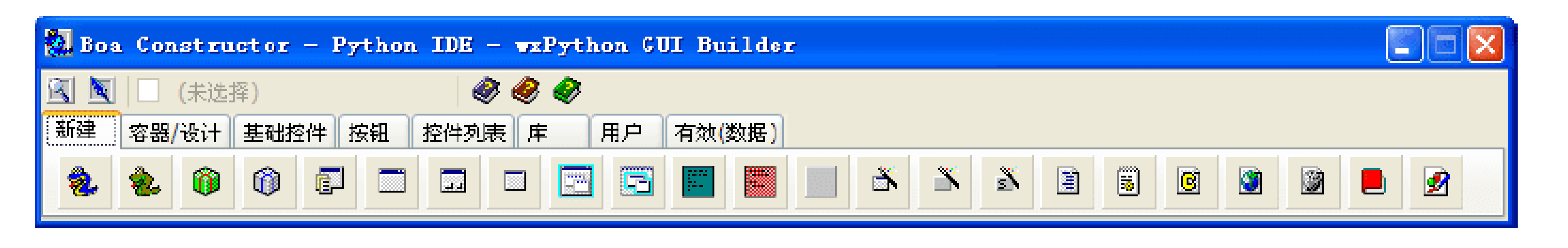


图 14-13　Boa-constructor程序截图

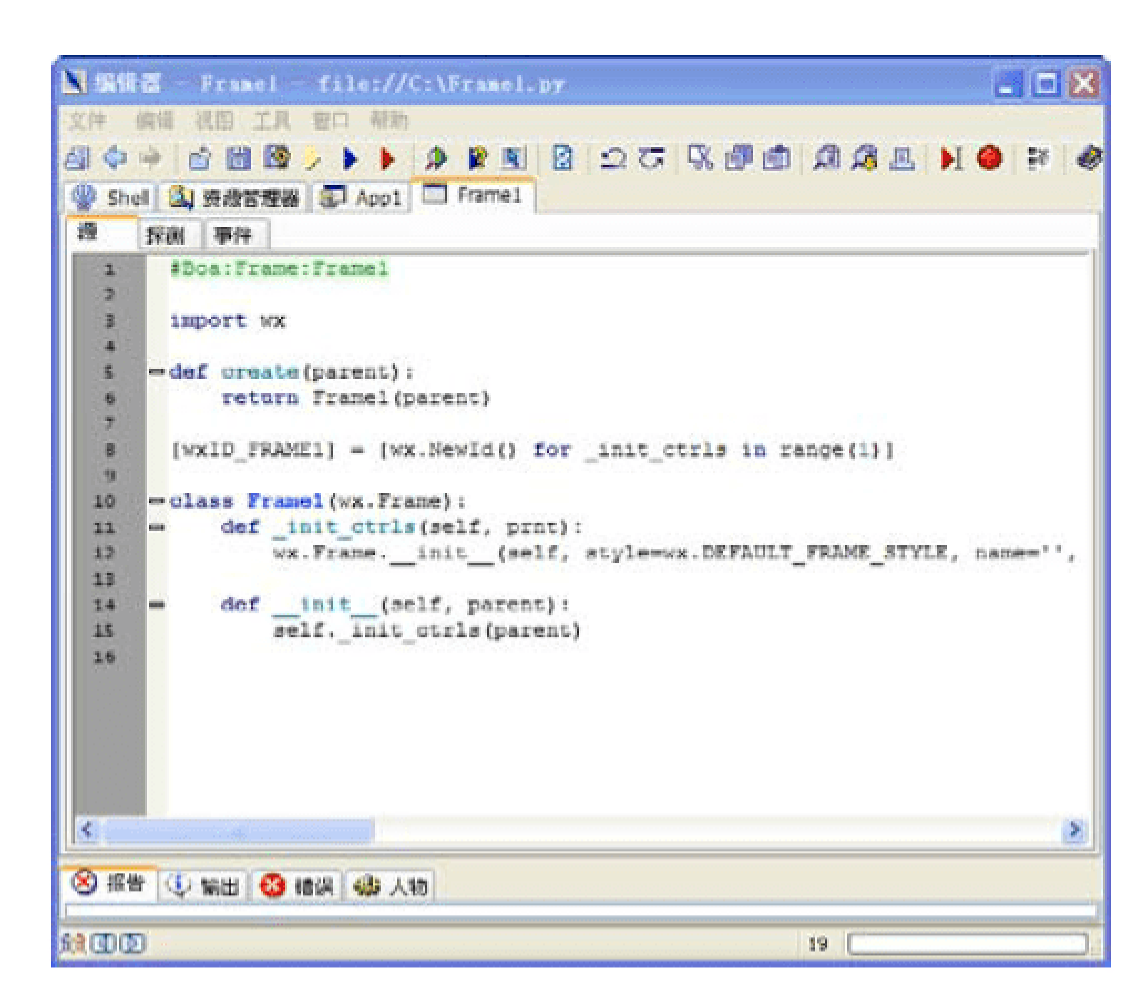



图 14-13　Boa-constructor程序截图(续)

14.2　wxPython的程序结构

为了能够加速 wxPython 下的界面开发，可以使用 GUI 设计工具。利用这些工具可以用可视化的方法来设计界面，从而减少编写代码的工作量。一个 wxPython 程序一般包含两个对象：应用程序对象和父窗口。其中，应用程序对象可以通过实例化 wx.App 来实现，父窗口可以通过 wx.Frame 来实现。本节将介绍一些开源 wxPython 开发工具和程序结构。

 视频教学：光盘/videos/14/ wxPython 的程序结构.avi　　长度：13 分钟

14.2.1　基础知识——wxPython应用程序的组成

每一个 wxPython 应用程序都有一个应用程序对象，这个应用程序对象拥有至少一个父窗口，这是 wxPython 程序必需的组成部分。另外，在应用程序对象中实现了一个事件循环处理，将处理窗口及其构建中的事件。

在 wxPython 中，可以通过实例化 wx.App 或者实例化从 wx.App 继承的子类来生成应用程序对象。此对象将会管理窗口系统中的事件并传递给特定的事件处理函数。

1. 使用wx.App生成应用程序对象

```
import wx
class MyFirstFrame(wx.Frame):
    def __init__ (self,parent):
        wx.Frame.__init__(self,parent,-1,'Hello World',size=(300,300))
        panel=wx.Panel(self)
        sizer=wx.BoxSizer(wx.VERTICAL)
        panel.SetSizer(sizer)
        txt=wx.StaticText(panel,-1,'Hello World')
        sizer.Add(txt,0,wx.TOP| wx.LEFT,100)
        self.Centre()
```

```
app=wx.App()
frame=MyFirstFrame(None)
frame.Show(True)
app.MainLoop()
```

在上述代码中，使用 app=wx.App()来生成应用程序对象。运行该段代码，生成 Hello World 窗口，如图 14-14 所示。

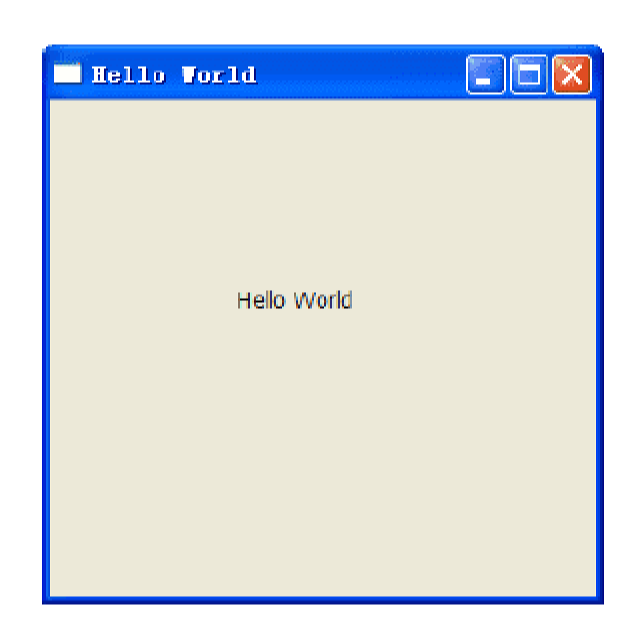


图 14-14　使用wx.App()生成应用程序对象

2. 使用wx.App继承子类的方式实现应用程序对象

```
import wx
# 省略 MyFirstFrame 类的定义，与上段代码相同
class MyApp(wx.App):
    def OnInit (self):
        self.frame=MyFirstFrame(None)
        self.frame.Show(True)
        return True
    def OnExit (self):
        pass
app=MyApp()
app.MainLoop()
```

在该段代码中，首先定义了一个 MyFirstFrame 类，该类继承自 wx.Frame，该类与上面代码相同。

不同的是，该段代码定义了一个 MyApp 类，此类继承自 wx.App 类。在该类中，首先定义了一个 OnInit()方法，这里的 OnInit()方法将在事件处理之前被 wxPython 系统调用。OnInit()方法是 wx.App 的子类中必需的。在此方法中实例化了前面定义的 MyFirstFrame 类，并调用 Show()方法将此窗口显示出来。



此方法最后返回值为 True，一般情况下这是 wxPython 应用程序所需要的。如果该方法返回的是 False，则应用程序执行完该方法之后会立即退出程序。很多情况下，因为应用程序所需要的资源缺失而无法继续运行。在 MyApp 类中又定义了 OnExit()方法，该方法则在事件处理结束之后执行，在这里并没有做任何操作。实际上，在这里可以加入退出时的清理工作。

在该段代码的最后，使用自定义的 MyApp 类构造了应用程序对象，而不是原来的 wx.App 类，最后调用 MainLoop()方法运行应用程序。运行结果与图 14-14 所示的窗口相同。



一般来说，OnExit()方法中的操作和 OnInit()方法中的操作相对应，例如在 OnInit()方法中执行了数据库的连接操作，那么在 OnExit()方法中可以断开数据库的连接。

14.2.2　基础知识——wxPython窗口的组成

窗口也称为框架(Frame)，它只是 wx.Frame 类的实例。在 wxPython 中，使用 wx.Frame 来创建 Frame 窗口。wx.frame 是所有 Frame 对象的父类。Frame 类的构造函数如下：

```
wx.Frame(window parent, int id, String title = EmptyString, Point pos =
DefaultPosition, Size size = DefaultSize, long style = DEFAULT_FRAME_STYLE,
String name = FrameNameStr)
```

其中：

- parent：表示当前窗口的父窗口。如果当前窗口是顶层窗口，则 parent=None，否则它的值为所属 Framen 的名字。
- id：一个新窗口的 ID 值。一般设为-1，这样 wxPython 会自动产生一个 ID。
- title：表示窗口的标题，即在标题栏上显示的内容。
- pos：它的值是一个 wx.Point 的对象，是一个元组，表示窗口创建时的位置坐标。
- size：它的值是一个 wx.Size 的对象，也是一个元组，表示窗口的大小。
- style：表示窗口的风格，窗口有多种风格，可以用 | 来组合多个风格。
- name：表示窗口的名称。

对父窗口而言，parent 参数设置为 None。Frame 将随着父窗口的销毁而销毁，也就是说，当关闭父窗口的时候，其子窗口同时被关闭。对于某些平台，将会限制子窗口在父窗口中的位置；pos 参数为窗口在显示屏幕中的初始化位置，默认为(-1,-1)，表示让窗口系统来选择。这里的位置是按照相对于屏幕左上角而言的；size 参数为窗口的初始化大小，默认为(-1,-1)，表示让窗口系统来决定这个尺寸。下面创建一个最简单的 Frame 窗口。

```
import wx
class MyFrame(wx.Frame):
   def __init__ (self):
       wx.Frame.__init__(self,None,-1,'自定义窗口',size=(400,200))
       self.Show()
app=wx.PySimpleApp()
myFrame=MyFrame()
app.MainLoop()
```

运行效果如图 14-15 所示。

对于每一个 wxPython 窗口，都有一个唯一的 ID 值。当然，这个唯一性是指在一个 Frame 之内，不同的 Frame 之间，ID 值可以重用。在应用程序中，可以使用 GetId()方法来获取当前 Frame 的 ID 值。请看下面的代码。

```
# 省略 MyFirstFrame 类的内容
class MyApp(wx.App):
   def OnInit (self):
       self.frame=MyFirstFrame(None)
       frameId=self.frame.GetId()
       print 'frameId=',frameId
```

```
        return True
# 省略使用自定义 MyApp 类构造应用程序对象，并调用 MainLoop()方法运行应用程序的代码
```



图 14-15　简单的Frame窗口

在这里定义的 OnInit()方法中，调用了 GetId()方法来获取当前 Frame 的 ID 值，并将其输出。运行该段代码，输出结果如下：

```
frameId= -201
```

当给 ID 参数传递的值为-1 时，wxPython 系统将会自动生成一个合适的 ID 值。实际上，-1 也是全局常量 wx.ID_ANY 的值。如果没有特殊的需求，可以采用这种方式来生成一个新的 ID 值，这样可以避免 ID 的冲突。另外一种方式是采用全局的 NewID()函数，同样可以保证 wxPython 系统生成的 ID 值在当前 Frame 中是唯一的。请看如下代码：

```
frame=wx.Frame.__init__(None,wx.NewId())
```

除此之外，每个窗口都有一定的样式，可以通过设置 style 参数来定义窗口部件的样式。一个窗口部件的显示表现在是否可以改变大小，是否可以最大化、最小化以及是否可以关闭等。实际上，窗口部件的样式可由很多样式元素组合而成。在 wxPython 中，这些样式元素通过一个常量来标识，而窗口部件的样式就是这些常量的组合。基本样式元素如表 14-2 所示。

表 14-2　窗口的样式

样　式	描　述
wx.DEFAULT_FRAME_STYLE	窗口的缺省风格，包含标题，可调节大小的边框，最大、最小化按钮，关闭按钮和系统菜单
wx.CAPTION	在框架上增加一个标题栏，它显示该框架的标题属性
wx.CLOSE_BOX	指示系统在框架的标题栏上显示一个关闭框，使用系统默认的位置和样式
wx.FRAME_ON_TOP	置顶窗口
wx.FRAME_SHAP ED	用这个样式创建的框架，可以使用 SetShape()方法创建一个非矩形的窗口

续表

样　式	描　述
wx.FRAME_TOOL_WINDOW	通过给框架一个比正常更小的标题栏，使框架看起来像一个工具框窗口。在 Windows 下，使用这个样式创建的框架不会出现在显示所有打开窗口的任务栏上
wx.MAXIMIZE_BOX	指示系统在框架的标题栏上显示一个最大化框，并使用系统默认的位置和样式
wx.MINIMIZE_BOX	指示系统在框架的标题栏上显示一个最小化框，并使用系统默认的位置和样式
wx.RESIZE_BORDER	给框架增加一个可以改变尺寸的边框
wx.SIMPLE_BORDER	没有装饰的边框。不能工作在所有平台上
wx.SYSTEM_MENU	增加系统菜单(带有关闭、移动、改变尺寸等功能)和关闭框到这个窗口上。在系统菜单中的改变尺寸和关闭功能的有效性依赖于 wx.MAXIMIZE_BOX, wx.MINIMIZE_BOX 和 wx.CLOSE_BOX 样式是否被应用
wx.FRAME_EX_META	如果在 Mac 中，这个属性用于是否显示金属风格
wx.FRAME_EX_CONTEXTHELP	是否有联机帮助按钮
wx.FRAME_FLOAT_ON_PARENT	窗口是否显示在最上层，与 wxSTAY_ON_TOP 不同，它必须有一个父窗口

14.2.3　实例描述

当移动腾讯 QQ 窗体靠近屏幕的边缘时，QQ 窗体会自动隐藏。如何实现这样的功能呢？当获取 QQ 窗体所在屏幕的位置之后，这个功能的实现并不难，只要获取鼠标所在的位置即可。下面就来使用 Python 语言实现获取鼠标当前位置的功能。

14.2.4　实例应用

【例 14-1】移动鼠标并获取鼠标当前位置。

(1) 新建 Python 文件，命名为 wx_1.py。

(2) 在 wx_1.py 文件中编辑程序代码。首先导入 wx 模块，然后定义 MyFrame 类，该类继承自 wx.Frame。在该类中定义两个方法，分别是__init__()和 OnMove()。代码如下：

```
import wx
class MyFrame(wx.Frame):
    def __init__(self):
        # 定义窗口的 ID、标题和大小
        wx.Frame.__init__(self,None,-1,'测试位置',size=(300,300))
        panel=wx.Panel(self,-1)
        # 触发鼠标移动事件
```

```
        panel.Bind(wx.EVT_MOTION,self.OnMove)
        wx.StaticText(panel,-1,"鼠标的当前位置为：",pos=(10,12))
        # 定义字体和文本框的位置
        self.posCtrl=wx.TextCtrl(panel,-1,"",pos=(120,10))
    def OnMove(self,event):
        pos=event.GetPosition()
        # 定义文本框的内容，获取鼠标所在位置
        self.posCtrl.SetValue("%s,%s"%(pos.x,pos.y))
```

(3) 编辑程序入口函数代码，构造 MyFrame 类对象，调用 Show()函数显示窗口，并调用 MainLoop()方法运行程序。代码如下：

```
if __name__=='__main__':
    app=wx.PySimpleApp()
    frame=MyFrame()
    frame.Show(True)
    app.MainLoop()
```

14.2.5　运行结果

运行 wx_1.py 文件，显示窗口。移动鼠标，当鼠标移动到工具栏的边缘时，显示的纵坐标为 0(窗口的左上角为原点，为(0,0))，如图 14-16 所示。当鼠标被移动到靠近窗口的左边框时，显示的横坐标为 0，如图 14-17 所示。当鼠标移动到窗口的中间部分时，显示当前鼠标所在的实际位置，如图 14-18 所示。

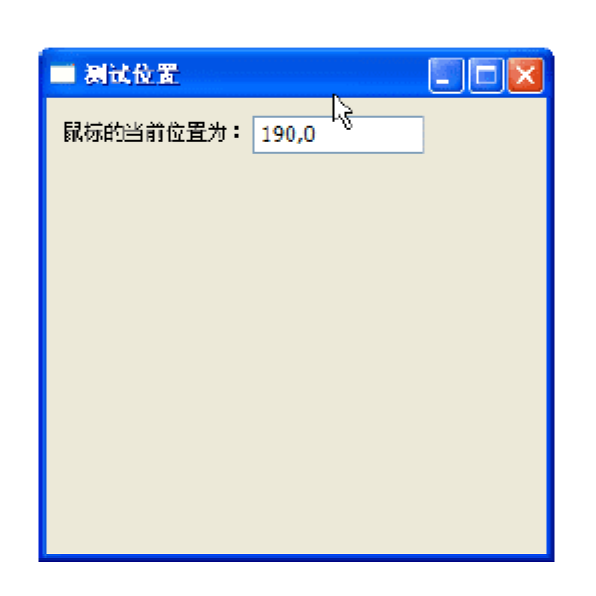


图 14-16　纵坐标为 0

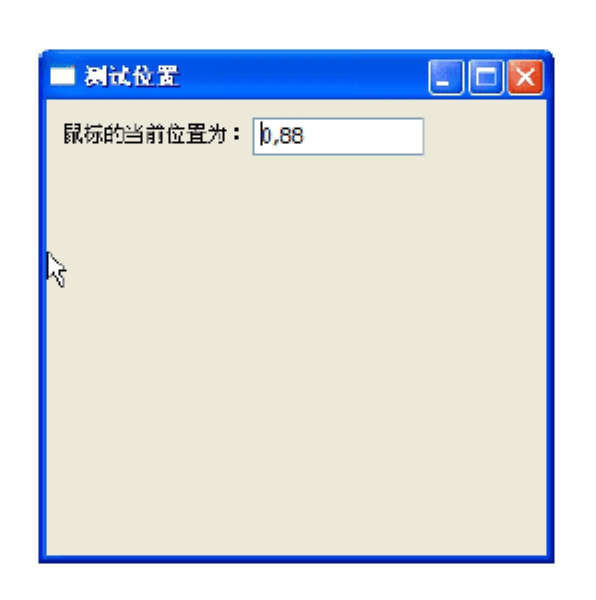


图 14-17　横坐标为 0



图 14-18　鼠标所在的当前位置

14.2.6　实例分析



源码解析

在该案例的__init__()方法中，使用 panel.Bind(wx.EVT_MOTION,self.OnMove)语句将鼠标移动事件绑定。其中，wx.EVT_MOTION 表示触发的事件是鼠标移动事件；sef.OnMove 表示要执行的函数为本类中的 OnMove()函数，同时在窗口中定义了一个显示鼠标横、纵坐标的文本框。接着在 MyFrame 类中定义了鼠标移动函数 OnMove()，使用 self.posCtrl.SetValue ("%s,%s"%(pos.x,pos.y))语句将获取的鼠标当前位置的横、纵坐标写入到文本框中。

14.3　wxPython的常用组件

在 wxPython 应用程序中，有些组件会经常见到。这些常用的组件包括对话框、工具栏和状态栏等。本节将对这些常用组件进行详细介绍。

 **视频教学：** 光盘/videos/14/ wxPython 的常用组件.avi　　　长度：12 分钟

14.3.1　基础知识——对话框

对话框是一种不含有最大、最小化按钮，并且一般不能改变形状大小的窗口。它包含按钮和各种选项，通过它们可以完成特定命令或任务。对话框是人与计算机交流的一种方式，用户对对话框进行设置，计算机就会执行相应的命令。wxPython 中的对话框为应用程序提供了与用户交互的信息，在 wxPython 中已经内置了不少对话框。下面简单介绍几种常见的对话框。

1．提示对话框

提示对话框通常用于帮助用户与计算机进行交互(例如退出应用程序时提示用户是否要退出、删除数据时提示用户是否确定删除等)。提示对话框由 wx.MessageDialog 类创建。wx.MessageDialog 类的构造函数如下：

```
wx.MessageDialog (Window parent, String message, String caption = MessageBoxCaptionStr,
long style = wxOK | wxCANCEL | wxCENTRE, Point pos = DefaultPosition)
```

其中，参数 parent 表示该 Dialog 的父窗口，如果没有则设为 None；参数 message 表示要在对话框中显示的提示信息；参数 caption 表示要显示在标题栏中的文本；参数 style 表示窗口风格(包括图标和按钮两种风格)，按钮可以包括 wxOK、wxCANCEL 和 wxCENTRE，当然也可以包括提示图标，例如 wx.ICON_ERROR；参数 pos 表示对话框的位置，一般为默认，当然也可以设置一个元组坐标位置。下面创建一个示例，介绍在 wxPython 中提示对话框的创建和使用。

```
import wx
class App(wx.App):
    def OnInit(self):
        dlg = wx.MessageDialog(None, '确定要退出程序吗？',
                            '程序退出提示', wx.YES_NO | wx.ICON_QUESTION)
        result = dlg.ShowModal()
        # 如果单击“是”按钮
        if result == wx.ID_YES:
            print '你单击了是按钮'
        dlg.Destroy()
        return True
if __name__ == '__main__':
    app = App(False)
    app.MainLoop()
```

在该段代码的 App 类中的 OnInit()方法中，创建了一个提示对话框。该对话框提示内容为“确定要退出程序吗？”，标题内容为“程序退出提示”，含有“是”和“否”两个按钮。接着使用 ShowModal()方法获取用户单击的按钮。如果单击“是”按钮，输出提示内容，否则释

放资源。在实例化 App()类时，传入了一个 Boolean 类型的参数 False，表示不再弹出窗口。运行该段代码，弹出提示对话框，如图 14-19 所示。如果用户单击“是”按钮，则在应用程序中输出“你单击了‘是’按钮”提示内容，否则释放资源，关闭对话框。

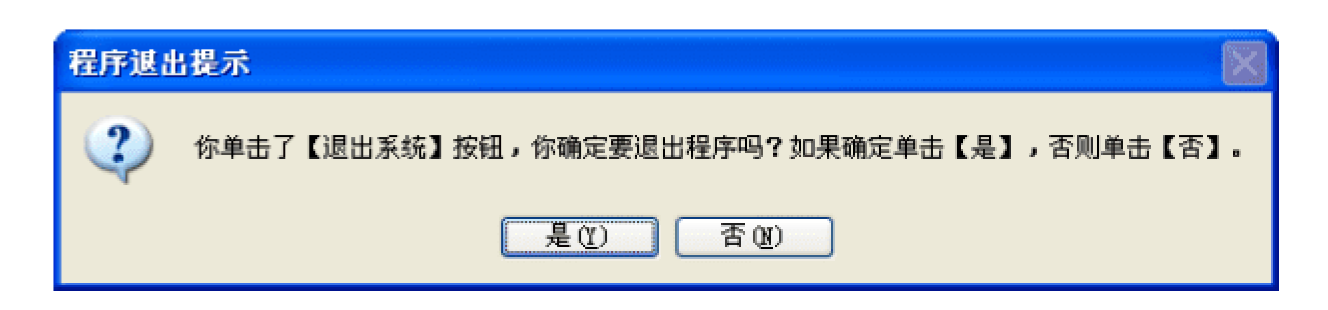


图 14-19　程序提示对话框

2．文本输入对话框

文本输入对话框用于返回输入字符串的值，文本输入对话框通常由一个文本输入框和 OK、Cancel 按钮组成。文本输入对话框可由 wx.TextEntryDialog 类创建，wx.TextEntryDialog 类的构造函数如下：

```
wx.TextEntryDialog(Window parent, String message, String caption =
GetTextFromUserPromptStr, String defaultValue = EmptyString, long style =
TextEntryDialogStyle, Point pos = DefaultPosition)
```

其中，参数 parent、message、caption、style 和 pos 参数与 wx.MessageDialog 类的构造函数中的参数含义相同。参数 defaultValue 表示输入框中初始显示的内容，默认为空。下面创建一个示例，具体介绍文本输入框的创建和使用。

```
import wx
class App(wx.App):
    def OnInit(self):
        dlg=wx.TextEntryDialog(None,"你最喜欢的一种语言？","一个问题","Python")
        if dlg.ShowModal()==wx.ID_OK:
            print dlg.GetValue()
        return True
if __name__ == '__main__':
    app = App(False)
    app.MainLoop()
```

在创建文本输入框时传入 4 个参数，分别为：父窗口、对话框中显示的文本内容、标题内容，以及输入框中初始化的文本内容，而对话框的风格和位置采用了默认值。运行该段代码，弹出文本输入对话框，如图 14-20 所示。单击 OK 按钮，在解释器中输出用户输入的值。

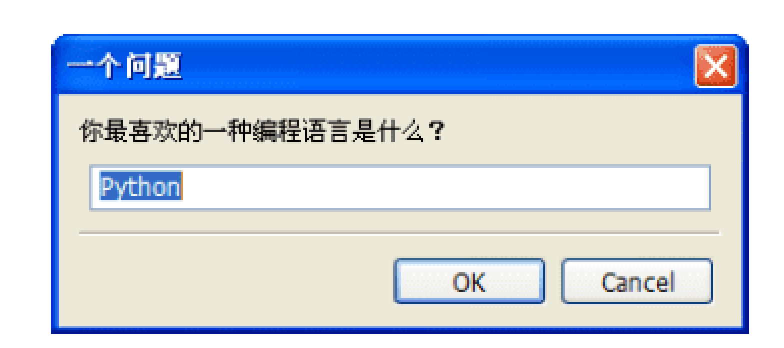


图 14-20　文本输入对话框

3. 选择文件对话框

选择文件对话框由 wx.FileDialog 类创建，该类的构造函数如下：

```
wx.FileDialog(Window parent, String message = FileSelectorPromptStr, String
defaultDir = EmptyString, String defaultFile = EmptyString, String wildcard =
FileSelectorDefaultWildcardStr, long style = FD_DEFAULT_STYLE, Point pos =
DefaultPosition)
```

其中，wildcard 表示文件类型的过滤字符串。该字符串由文件类型名称、文件后缀两部分组成(例如 Python source:*.py，表示文件类型为 Python 源文件，文件后缀名为 py)。如果需要打开多种类型的文件，则需要在多种过滤字符串之间用 | 分隔，例如 Python source(*.py) | *.py | All files(*.*) | *.*，表示打开 Python 源文件和所有文件。下面创建一个示例，具体介绍如何创建一个特殊对话框。

```
import wx
import os
class App(wx.App):
    def OnInit(self):
        fileFilter="Python source(*.py)|*.py|All files(*.*)|*.*"
        dlg=wx.FileDialog(None, "选择文件",os.getcwd(),"",fileFilter,wx.OPEN)
        dlg.ShowModal()
        dlg.Destroy()
        return True
if __name__ == '__main__':
    app=App()
    app.MainLoop()
```

在 OnInit()方法中，创建了“选择文件”对话框。该对话框将要打开显示当前目录中的文件，并对可打开的文件类型进行过滤，文件类型下拉列表框中提供 py 为后缀的文件和所有文件两种类型。其中 os.getcwd()表示获取当前文件所在的路径。运行程序，弹出“选择文件”对话框，如图 14-21 所示。

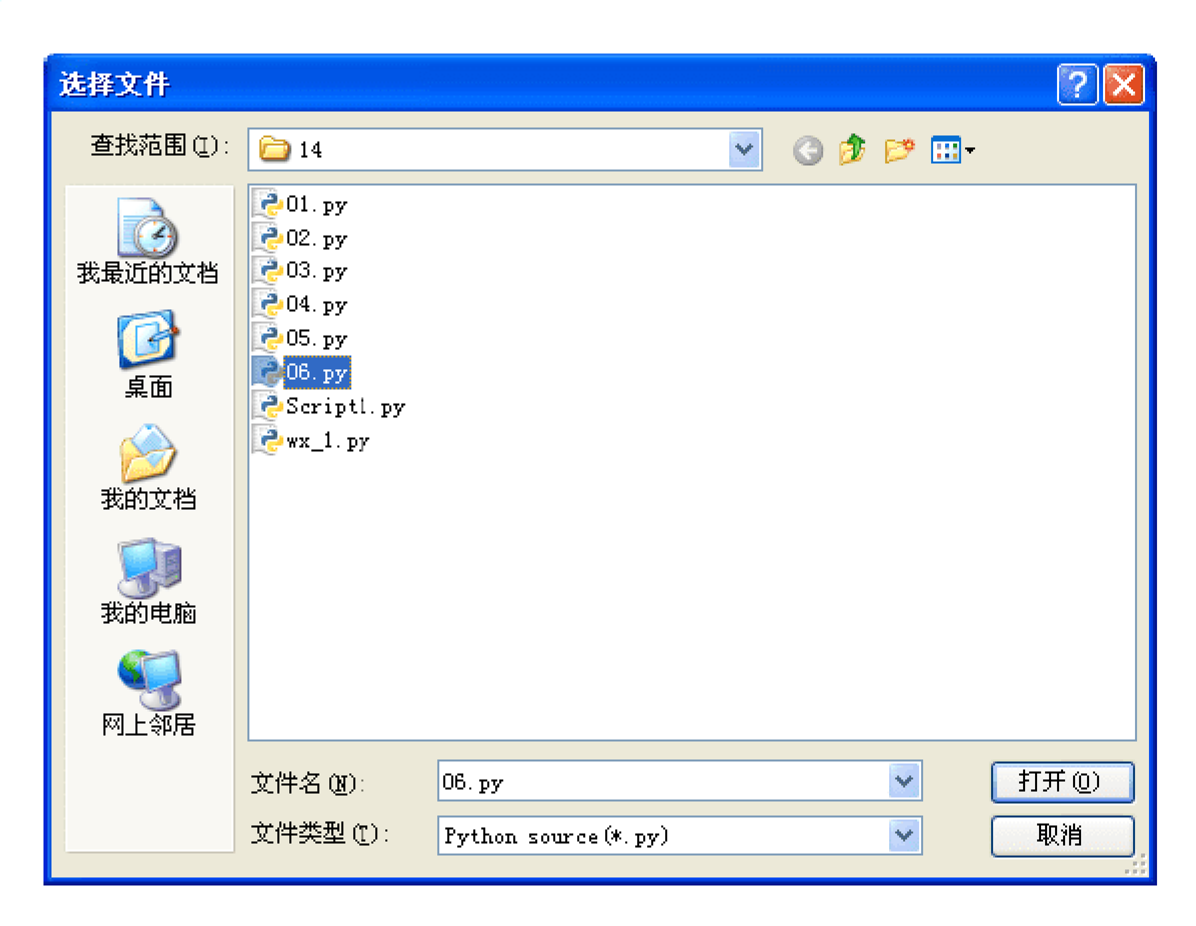


图 14-21　“选择文件”对话框

14.3.2 基础知识——工具栏

工具栏提供了几种常用的操作按钮，比如向用户表中添加一条数据记录时要用到的“添加”按钮、删除一条数据记录时要用到的“删除”按钮等。在 wxPython 中，可以通过调用 Frame 的 CreateToolBar()方法来生成，并且需要使用 AddSimpleTool()方法向工具栏中添加按钮。下面创建一个示例，具体介绍如何创建一个工具栏。

```
import wx
import wx.py.images as images
class ToolbarFrame(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, '工具栏',
                size=(500, 300))
        panel = wx.Panel(self)
        panel.SetBackgroundColour('White')
        toolbar = self.CreateToolBar()              # 创建工具栏
        toolbar.AddSimpleTool(wx.NewId(), images.getPyBitmap(),
                "New", "Long help for 'New'")       # 给工具栏增加一个按钮选项
        toolbar.Realize()                           # 准备显示工具栏
if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = ToolbarFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

在窗口中仅仅有一个工具栏，此工具栏使用 CreateToolBar()方法创建。为了在工具栏中添加新的按钮，可以使用工具栏对象的 AddSimpleTool()方法，并在此方法中指定工具栏要显示的图片，然后使用工具栏对象的 Realize()方法显示工具栏。运行该段程序代码，弹出含有工具栏的窗口，当把鼠标放到工具栏的按钮上时，显示 New，如图 14-22 所示。

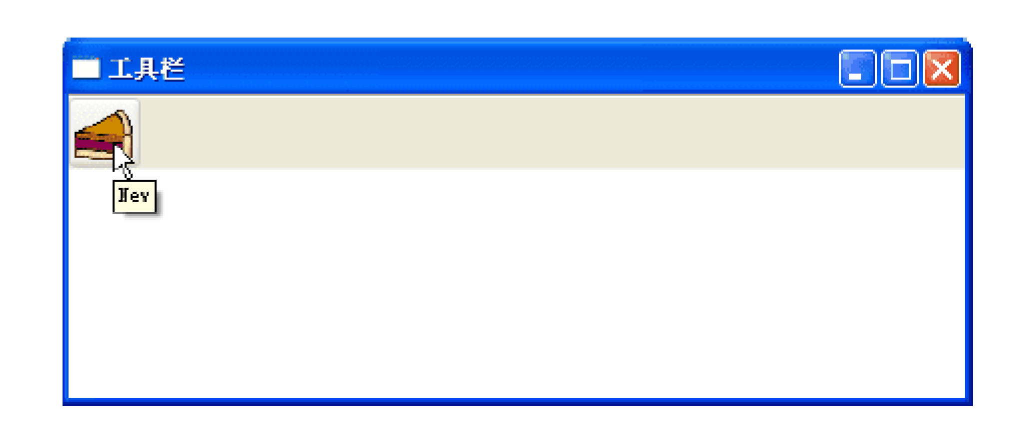


图 14-22 含有工具栏的窗口

14.3.3 基础知识——状态栏

状态栏是指在每个窗口、程序操作界面的最底端，用来显示特定的信息。在 wxPython 中，可以使用 CreateStatusBar 方法来创建状态栏。下面创建一个示例，具体介绍如何创建状态栏。

```
import wx
class StatusbarFrame(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, '状态栏',
                size=(500, 200))
        panel = wx.Panel(self)
```

```
        panel.SetBackgroundColour('White')
        statusbar = self.CreateStatusBar()        # 创建状态栏
        statusbar.SetStatusText("状态栏信息")      # 给状态栏添加显示信息
if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = StatusbarFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

该段代码首先使用 CreateStatusBar 加入了一个状态栏，随后调用了 SetStatusText()方法来设置状态栏的信息。运行该段代码，弹出一个含有状态栏的窗口，如图 14-23 所示。

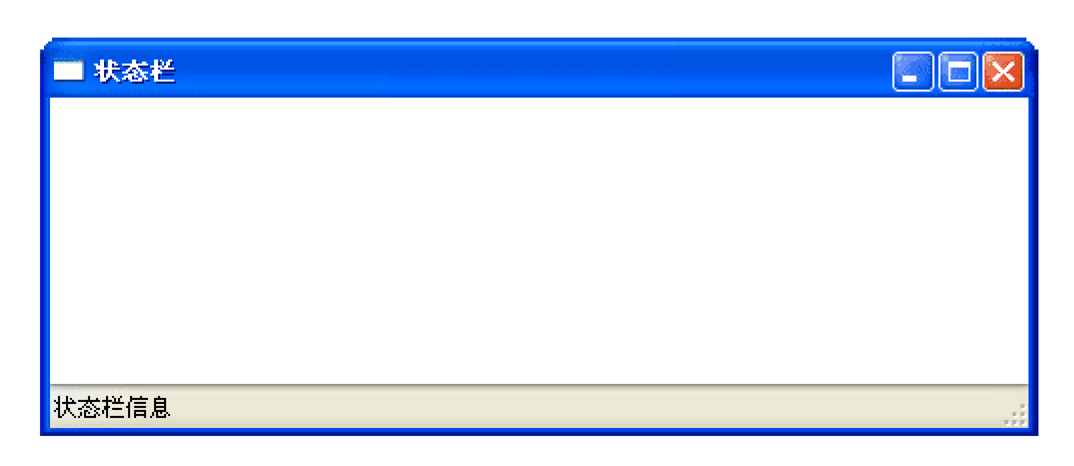


图 14-23　含有状态栏的窗口

14.3.4　实例描述

工具栏提供了常用操作的快捷键按钮，比如 Microsoft Office World 的工具栏提供了新建空白文档、打开、保存、电子邮件等常用的操作按钮，而状态栏则显示了该 World 文档的一些基本信息，比如当前是第几页、总共几页等。在程序开发中，工具栏显得尤为重要。当单击退出程序按钮时，系统会提示用户是否确定退出，以防用户误操作而导致一些不必要的麻烦。下面探究一下工具栏、状态栏和对话框的魅力所在。

14.3.5　实例应用

【**例 14-2**】实现单击工具栏图标弹出提示对话框的功能。

(1) 新建 Python 文件，命名为 wx_2.py。

(2) 在 wx_2.py 文件中创建 ToolbarFrame 类。在该类中的__init__()初始化方法中创建工具栏和状态栏，并绑定事件(绑定事件在后面的章节中将会详细讲解)。当单击工具栏上的退出程序图标时，触发退出事件即 OnExit()方法；当用户单击工具栏上的删除图标时，触发删除事件即 OnDelete()方法。ToolbarFrame 类的代码内容如下：

```
import wx
class ToolbarFrame(wx.Frame):
    def __init__(self, parent, id):
        wx.Frame.__init__(self, parent, id, '工具栏', size=(600, 300))
        panel = wx.Panel(self)
        panel.SetBackgroundColour('White')
        toolbar = self.CreateToolBar()                    #创建工具栏
        toolbar.AddSimpleTool(wx.NewId(),wx.Image( 'ico/xin.png',
wx.BITMAP_TYPE_PNG ).ConvertToBitmap(), "New")      #给工具栏增加一个工具
```

```
        toolbar.AddSimpleTool(wx.NewId(),wx.Image( 'ico/save.png',
wx.BITMAP_TYPE_PNG ).ConvertToBitmap(), "Save")    #给工具栏增加一个工具
        toolbar.AddSimpleTool(wx.ID_DELETE,wx.Image( 'ico/abc.png',
wx.BITMAP_TYPE_PNG ).ConvertToBitmap(), "Delete")  #给工具栏增加一个工具
        toolbar.AddSimpleTool(wx.ID_EXIT, wx.Image( 'ico/exit.png',
wx.BITMAP_TYPE_PNG ).ConvertToBitmap(), "Exit")
        toolbar.SetToolBitmapSize(wx.Size (10, 10))
        toolbar.Realize()                               # 准备显示工具栏

        statusbar = self.CreateStatusBar()              # 创建状态栏
        statusbar.SetStatusText("版权所有：河南省郑州市汇智科技有限公司 技术支持：汇智
科技全体员工")                                          # 给状态栏添加显示信息
        #绑定事件，用 id 来绑定事件
        self.Bind(wx.EVT_TOOL, self.OnExit, id=wx.ID_EXIT)
        self.Bind(wx.EVT_TOOL,self.OnDelete,id=wx.ID_DELETE)
```

(3) 当用户单击工具栏上的退出图标时，弹出提示对话框，提示用户是否确定退出程序。如果用户单击“是”按钮，则退出程序，否则关闭该提示框。接着在 ToolbarFrame 类中编辑 OnExit()方法，代码如下：

```
def OnExit(self,event):
        dlg = wx.MessageDialog(None, '确定要退出程序吗？',
                    '程序退出提示', wx.YES_NO | wx.ICON_QUESTION)
        result = dlg.ShowModal()
        # 如果单击“是”按钮
        if result == wx.ID_YES:
            self.Close(True)
        dlg.Destroy()
```

(4) 当用户单击工具栏上的删除图标时，弹出提示对话框，提示用户是否确定删除该数据时。如果确定则弹出一个只含有“确定”按钮的对话框，表示已经删除该数据，否则关闭提示对话框。下面首先来编辑 MyDialog 类，并使用 Show()方法显示该对话框。使用 Show()方法显示的对话框称为非模式对话框，该对话框的窗口可以失去焦点，以便对主程序窗口进行其他操作。MyDialog 的完整代码如下：

```
class MyDialog(wx.Dialog):
    def __init__(self,parent,id,title):
        wx.Dialog.__init__(self,parent,id,title,size=(200,200))
        self.panel=wx.Panel(self)
        # 向对话框中添加按钮
        self.OkBtn=wx.Button(self,10,'确定',pos=(50,100),size=(80,30))
        # 绑定事件
        self.Bind(wx.EVT_BUTTON,self.CloseDlg,self.OkBtn)
        self.Show()
    def CloseDlg(self,event):
        self.Close()
```

在这里，该类继承自 wx.Dialog，并非 wx.Frame，即表示该类生成的不是框架而是一个对话框。接着使用 self.OkBtn=wx.Button(self,10,'确定',pos=(50,100),size=(80,30))语句定义了在该对话框中有一个“确定”按钮，并指定了该按钮的位置和大小。之后使用 Bind()方法绑定该按钮的触发事件为 CloseDlg()方法。

在 ToolbarFrame 类中编辑 OnDelete()方法，当用户单击“是”按钮时，调用 MyDialog 类，

否则关闭提示对话框。

```
def OnDelete(self,event):
        dlg = wx.MessageDialog(None, '确定要删除该条数据吗？',
                  '删除数据提示', wx.YES_NO | wx.ICON_QUESTION)
        result = dlg.ShowModal()
        # 如果单击“是”按钮
        if result == wx.ID_YES:
            mydialog=MyDialog(parent=None,id=-1,title='删除')
        dlg.Destroy()
```

(5) 最后调用 ToolbarFrame 类，创建含有工具栏和状态栏的窗口。代码如下：

```
if __name__ == '__main__':
    app = wx.PySimpleApp()
    # 实例化 ToolbarFrame 类，并将 parent 和 id 参数传递过去
    frame = ToolbarFrame(parent=None, id=-1)
    frame.Show()
    app.MainLoop()
```

14.3.6　运行结果

运行 wx_2.py 文件，出现一个含有工具栏和状态栏的窗口，如图 14-24 所示。当单击工具栏上的第三个按钮，即删除数据按钮时，弹出提示用户是否确定删除数据的对话框，如图 14-25 所示。



图 14-24　含有工具栏和状态栏的窗口



图 14-25　提示用户是否确定删除对话框

当用户单击该对话框中的“是”按钮时，弹出只包含一个“确定”按钮的对话框，如图 14-26 所示。当用户单击窗口中的第四个按钮，即单击退出程序按钮时，弹出提示用户是否确定退出程序的对话框，如图 14-27 所示。如果单击“是”按钮，则退出程序，窗口关闭，否则关闭提示对话框。



图 14-26　弹出删除成功对话框



图 14-27　提示用户是否确定退出对话框

14.3.7　实例分析



源码解析

在该案例中创建了两个类：一个用于显示窗口的类，另一个用于显示只包含有一个“确定”按钮的对话框类。对话框的创建和使用与 Frame 相似，不同的是对话框表示一次信息交换的活动，当交换完成后单击对话框中的按钮则将关闭该对话框，对话框只是应用程序生命周期的一部分。在表示对话框的类 MyDialog 中，使用了 Show()方法来显示对话框，表示该对话框并非是模式对话框，如果使用 ShowModal()方法显示对话框，则此时的对话框称为模式对话框，用户必须关闭对话框后，才能对主程序窗口进行操作。

14.4　wxPython的基本组件

wxPython 的基本组件包括文本框、按钮、单选框、复选框、列表等。wxPython 的列表控件非常丰富，可以定义列表框和下拉列表框，也可以创建可编辑的下拉列表组件。布局管理器统一管理界面中的组件，为组件的排列布局提供了简单的方式，而不用程序设计者显式指定各个组件的位置。wxPython 提供了 Grid、Flex、Box 等布局管理器，根据不同的场合使用相应的布局管理器可以美化应用程序。本节将详细介绍 wxPython 中的基本组件和几种常用的布局管理器。

视频教学：光盘/videos/14/文本框.avi　　长度：7 分钟

视频教学：光盘/videos/14/wxPython 其他组件.avi　　长度：18 分钟

视频教学：光盘/videos/14/Sizers 布局组件.avi　　长度：10 分钟

14.4.1　基础知识——文本框

文本框控件是应用程序中最常用的一种控件，在 wxPython 中的 StaticText 类表示静态文本

框，而 TextCtrl 类表示输入文本框。

1. 静态文本框

在 wx.StaticText 类中，可以使用 StaticText 类来设置文本框的大小、颜色、对齐方式和字体等属性。简单的静态文本控件可以包含多行文本，但是不能处理多种字体或样式。StaticText 类在 wx 包中，因此在使用静态文本框时，需要先将 wx 包导入到文件中。静态文本框的样式可以通过构造函数的参数来设置，静态文本框的构造函数如下：

```
wx.StaticText(Window parent, int id, String label = EmptyString, Point pos =
DefaultPosition, Size size = DefaultSize, long style = 0,String name =
StaticTextNameStr)
```

其中：

- parent：静态文本框控件的父容器。
- id：标识当前的静态文本框控件。
- label：静态文本框框中输入的字符串。
- pos：静态文本框左上角的坐标。
- size：一个有两个元素的元组，两个元素分别表示静态文本框的长度和宽度。
- style：静态文本框中字符串的对齐方式，其中 wx.ALIGN_CENTER 表示静态文本位于静态文本控件的中心，wx.ALIGN_LEFT 表示静态文本在窗口部件中左对齐(默认样式)，wx.ALIGN_RIGHT 表示静态文本在窗口部件中右对齐。
- name：当前静态文本框的名称，可用于控件的查找。

下面创建一个示例，具体介绍如何创建一个静态文本框。

```
import wx
class StaticTextFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'Static Text Example',size=(300,200))
        panel=wx.Panel(self,-1)
        # 创建静态文本框，其长度为 200 像素，宽度为 90 像素，位于控件的中心
        text=wx.StaticText(panel,-1,'静态文本框',(70,50),(200,90),wx.ALIGN_CENTER)
        # 创建字体类型
        font=wx.Font(14,wx.DEFAULT,wx.ITALIC,wx.NORMAL,True)
        text.SetFont(font)                   # 设置字体
        text.SetForegroundColour('red')      # 设置静态文本框的前景色
        text.SetBackgroundColour('yellow')   # 设置静态文本框的背景色
if __name__=='__main__':
    app=wx.PySimpleApp()
    frame=StaticTextFrame()
    frame.Show()
    app.MainLoop()
```

在该段代码中，首先调用 StaticText 类的构造函数创建了静态文本框，其文本的内容为“静态文本框”，文本的左上角位置为(70,50)，文本的长度为 200，宽度为 90。接着通过 Font 类设置了静态文本框中文字的字体，分别使用 SetForegroundColour()和 SetBackgroundColour()方法设置了静态文本框的前景色和背景色。运行效果如图 14-28 所示。



图 14-28　静态文本框

技术文档	Font类

通过 Font 类可以设置静态文本框中文字的字体，Font 类的构造函数如下：

wx.Font(int pointSize, int family, int style, int weight, bool underline = Fale, String face = EmptyString, int encoding = FONTENCODING_DEFAULT)

其中，参数 pointSize 用于设置文字的字号；参数 family 用于设置文字的字体类型；参数 style 设置文字的样式，如斜体、粗体等；参数 weight 设置文字的大小；参数 underline 表示是否在文字下方输出下划线，该参数的默认值为不输出下划线；参数 face 表示文字的外观，该参数为可选参数；参数 encoding 用于设置文字的编码。

2. 输入文本框

wxPython 提供了 TextCtrl 类来创建输入文本框，TextCtrl 类的构造函数的参数与 StaticText 类似。TextCtrl 类的构造函数如下：

```
TextCtrl(Window parent, int id, String value = EmptyString, Point pos =
DefaultPosition, Size size=DefaultSize, long style =0, Validator
validator=DefaultValidator, String name = TextCtrlNameStr)
```

可以看出，TextCtrl 类比 StaticText 类的构造函数多提供了一个 Validator 参数，该参数是 wx.Validator 类型的，用于过滤数据，使文本框只能接受规定类型的数据。

参数 style 用来控制文本框的对齐方式、显示方式以及文本框的输入方式。TextCtrl 控件的样式规则如表 14-3 所示。

表 14-3　输入文本框的样式

样　式	描　述
wx.TE_CENTER	输入文本框中的内容居中显示
wx.TE_LEFT	输入文本框中的字符串左对齐
wx.TE_RIGHT	输入文本框中的字符串右对齐
wx.TE_NOHIDESEL	输入文本框内容高亮显示
wx.TE_PASSWORD	密码输入框，隐藏输入的文字
wx.TE_PROCESS_ENTER	在输入文本框中按下回车键时，触发相关事件
wx.TE_PROCESS_TAB	当按下 Tab 键切换控件后，触发相关事件

续表

样　式	描　述
wx.TE_READONLY	设置输入文本框为只读状态
wx.TE_MULTILINE	设置输入文本框为多行文本框
wx.TE_RICH	设置输入文本框的文字可以使用字体、颜色和样式
wx.TE_RICH2	最新版的 TE_RICH
wx.TE_AUTO_URL	如果 TE_RICH 或 TE_RICH2 中存在 URL，该样式可以响应 URL 的单击
wx.TE_LINEWRAP	如果内容太长，将以字符为界换行
wx.TE_WORDWRAR	如果内容太长，将以单词为界换行



默认情况下，Enter 键事件和 Tab 键事件由对话框管理。如果要使 TextCtrl 控件响应 Enter 键事件或 Tab 键事件，则需要在 style 中定义相关的样式，然后再绑定事件并编写事件处理函数。

下面创建一个包含输入文本框、密码输入文本框和多行文本输入框的实例。当输入用户名后，按 Tab 键切换至密码输入框时，将弹出对话框显示用户名；当输入用户名和密码之后，在密码输入框中按 Enter 键时，将弹出对话框并显示用户名和密码。代码如下：

```
import wx
class MultiTextFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self,None,-1,"用户注册界面",
                          size=(500,400))
        panel=wx.Panel(self,-1)
        #创建一个静态文本输入框
        labName=wx.StaticText(panel,-1,'用户名',pos=(50,10))
        #创建一个普通文本输入框
self.inputText=wx.TextCtrl(panel,-1,'',pos=(120,10),size=(150,-1),style=wx.
TE_LEFT|wx.TE_PROCESS_TAB)
        self.inputText.SetInsertionPoint(0)
        labPwd=wx.StaticText(panel,-1,'密码',pos=(50,50))
        #创建一个密码输入框，当在密码输入框中按下回车键后触发事件
self.pwdText=wx.TextCtrl(panel,-1,'',pos=(120,50),size=(150,-1),style=wx.TE
_PASSWORD|wx.TE_PROCESS_ENTER)
        #绑定事件
        self.Bind(wx.EVT_TEXT_ENTER,self.OnLostFocus,self.pwdText)
        #普通多行文本输入框
        multiLabel=wx.StaticText(panel,-1,"网络服务协议:",pos=(40,90))
        multiText=wx.TextCtrl(panel,-1,"本协议由您与汇智科技有限公司共同缔结，…… ",
                              pos=(120,90),
                              size=(300,100),
                              style=wx.TE_MULTILINE)

        #丰富样式的多行文本输入框
        richLabel=wx.StaticText(panel,-1,"网络服务协议:",pos=(40,210))
        richText=wx.TextCtrl(panel,-1,
                             "本协议由您与汇智科技有限公司共同缔结，……",
                             pos=(120,210),
                             size=(300,100),
```

```
                          #创建丰富文本控件
                          style=wx.TE_MULTILINE|wx.TE_RICH2)
        #设置 richText 控件的文本样式
        richText.SetStyle(2,6,wx.TextAttr("white","black"))
        points=richText.GetFont().GetPointSize()
        #创建一个字体样式
        f=wx.Font(points+3,wx.ROMAN,wx.ITALIC,wx.BOLD,True)
        #用创建的字体样式设置文本样式
        richText.SetStyle(8,14,wx.TextAttr("blue",wx.NullColor,f))
    def OnLostFocus (self,event):
wx.MessageBox('username:%s,password:%s'%(self.inputText.GetValue(),self.pwd
Text.GetValue()),'admin')
def main():
    app=wx.PySimpleApp()
    frame=MultiTextFrame()
    frame.Show(True)
    app.MainLoop()
if __name__=="__main__":
    main()
```

该段代码首先创建一个普通输入文本框，接着创建一个密码输入框，然后创建一个多行文本输入框，最后创建一个丰富样式的多行文本输入框，并设置了 richText 的字体样式。运行效果如图 14-29 所示。当输入用户名和密码之后，在密码输入框中按回车键，则弹出显示用户名和密码的对话框，如图 14-30 所示。

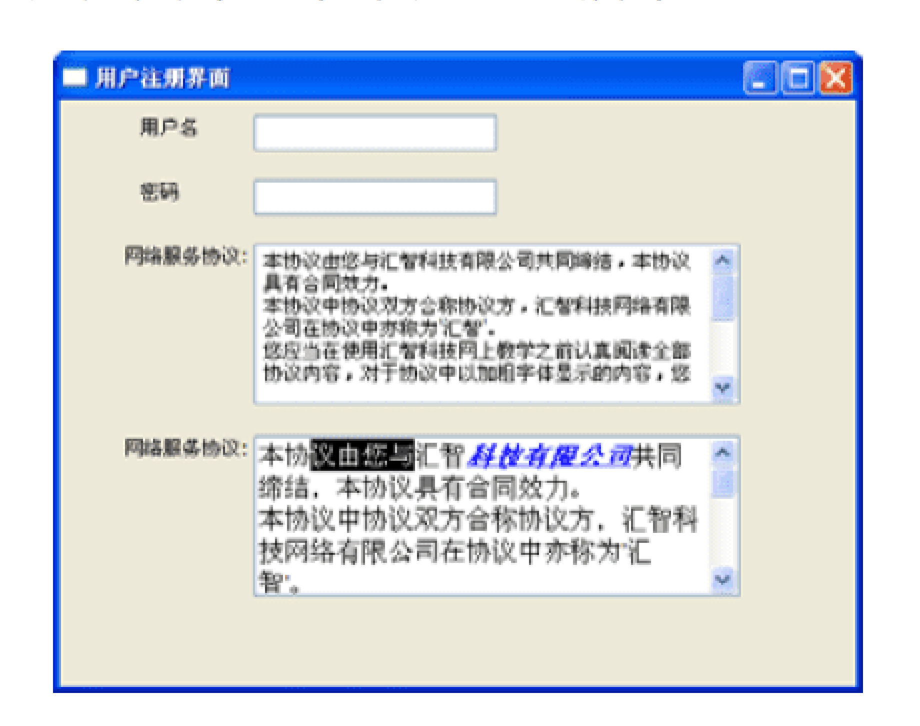


图 14-29　含有多种样式的文本框



图 14-30　在密码输入框中按下回车键

14.4.2　基础知识——按钮控件

按钮控件是应用程序中经常用到的控件之一，本节将介绍按钮控件的创建以及位图控件的使用方法。

1．普通按钮

普通按钮可使用 wx.Button 类来创建，Button 类的构造函数如下：

```
wx.Button(Window parent, int id, String label = EmptyString, Point pos =
DefaultPosition, Size size = DefaultSize, long style=0, Validator
validator=DefaultValidator, String name = ButtonNameStr)
```

Button()类的构造函数中的参数与 TextCtrl 类相同，这里不再累赘。下面做一个示例，具

体介绍普通按钮的创建。

```
import wx
class ButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, '普通按钮', size=(300, 100))
        panel = wx.Panel(self, -1)
        #创建一个普通按钮
        self.button = wx.Button(panel, -1, "确定", pos=(50, 20),size=(100,30))
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)     #绑定按钮事件
        self.button.SetDefault()      #设置为默认值
    def OnClick(self, event):       #当触发按钮事件后，按钮上的文字变为：你已经点过了
        self.button.SetLabel("你已经点过了")
if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = ButtonFrame()
    frame.Show()
    app.MainLoop()
```

在该段代码中，使用 Button 类的构造函数创建了 Button 对象，并使用 Bind()方法绑定了该按钮的触发事件，然后使用 SetDefault()方法将 Button 设置为默认按钮。该段代码运行的效果如图 14-31 所示，当单击“确定”按钮后，按钮上的文字变为“你已经点过了”，如图 14-32 所示。



图 14-31　普通按钮



图 14-32　单击后文字改变

2. 位图按钮

wxPython 提供了专门的位图控件，wx.BitmapButton 类可以用来创建位图按钮，该类的构造函数如下：

```
wx.BitmapButton(Window parent, int id, Bitmap bitmap=wxNullBitmap, Point pos=
DefaultPosition, Size size = DefaultSize, long style = BU_AUTODRAW, Validator
validator = DefaultValidator, String name = ButtonNameStr)
```

创建位图按钮之前需要定义 Image 类型的对象，并把该对象转换为 Bitmap 类型，然后再创建 BitmapButton 类型的按钮，并关联 Bitmap 的实例化对象。下面创建一个示例，具体介绍如何使用 GIF 格式的图片创建位图按钮。

```
import wx
class BitmapButtonFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1, '位图按钮', size=(400, 200))
        panel = wx.Panel(self, -1)
        jpg = wx.Image("ico/02.jpg", wx.BITMAP_TYPE_JPEG).ConvertToBitmap()
        self.button = wx.BitmapButton(panel, -1, jpg, pos=(90, 20),size=(200,95))
        self.Bind(wx.EVT_BUTTON, self.OnClick, self.button)
        self.button.SetDefault()
    def OnClick(self, event):
        self.Destroy()
```

```
if __name__ == '__main__':
    app = wx.PySimpleApp()
    frame = BitmapButtonFrame()
    frame.Show()
    app.MainLoop()
```

在该段代码中，首先使用 Image 类的构造函数生成 Image 类对象，并使用该对象的 ConvertToBitmap()方法将 JPEG 格式的图片转换为 Bitmap 类型，然后使用 wx.BitmapButton()类的构造函数创建了一个位图按钮，并使用 Bind()方法绑定该按钮的触发事件。该段代码运行的效果如图 14-33 所示。



图 14-33 位图按钮

14.4.3 基础知识——单选按钮

在某个网站注册时，会注意到“性别”有两个选项：男和女。如果不允许用户选择多个选项，可以使用表单元素的单选按钮对象。单选按钮对象用于一组互相排斥的值，也就是用户只能从选项列表中选择一项。当出现多组选项时，通常可以把同一类选项作为一组来使用，这种控件称为单选按钮分组。本节将介绍如何创建单选按钮及如何将单选按钮分组。

1．创建单选按钮

单选按钮通常成组出现，用户每次操作只能选择其中一个单选按钮。使用 wx.RadioButton 类可以创建单选按钮控件，RadioButton 类的构造函数如下：

```
wx.RadioButton(Window parent, int id, String label = EmptyString, Point pos =
DefaultPosition, Size size = DefaultSize, long style = 0, Validator =
DefaultValidator, String name = RadioButtonNameStr)
```

RadioButton 类每次只创建一个单选按钮控件。下面创建两个单选按钮控件，分别代表男和女。

```
import wx
class RadioFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'单选按钮',size=(200,150))
        panel=wx.Panel(self,-1)
        #创建一个文本为“男”的单选按钮
        radioMale=wx.RadioButton(panel,-1,'男',pos=(20,20))
        #创建一个文本为“女”的单选按钮
        radioFemale=wx.RadioButton(panel,-1,'女',pos=(20,40))
if __name__=='__main__':
    app=wx.PySimpleApp()
    frame=RadioFrame()
    frame.Show()
```

```
    app.MainLoop()
```

单选按钮的创建很简单，只要使用 wx.RadioButton 类的构造函数传入相应的参数即可。该段代码运行后的效果如图 14-34 所示。

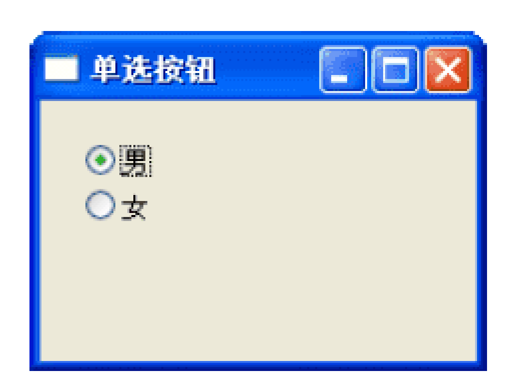


图 14-34　单选按钮

2．单选按钮的分组

在 wxPython 中还可以使用 RadioBox 类来创建单选按钮，该类用于创建一组单选按钮控件，这组单选按钮将显示在一个矩形中。RadioBox 类的构造函数如下：

```
wx.RadioBox(Window parent, int id, String label = EmptyString, Point pos =
DefaultPosition, Size size = DefaultSize, wxArrayString choices =
wxPyEmptyStringArray, int majorDimension = 0, long style = RA_HORIZONTAL,
Validator validator = DefaultValidator, String name = RadioBoxNameStr)
```

其中：

- choices：一个字符串列表，表示 RadioBox 中各个 RadioButton 控件的标签名称。
- majorDimension：一个整型数字，根据参数 style 及此数字来排列 RadioButton 控件。
- style：设置 RadioButton 控件的排列方式。其中 wx.RA_SPECIFY_COLS 表示按照列进行排列，wx.RA_SPECIFY_ROWS 表示按照行进行排列。

下面使用 RadioBox 控件对列表中的文本进行分组，每个文本对应一个单选按钮。

```
import wx
class RadioBoxFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self, None, -1,'单选按钮的分组',size=(400,200))
        panel=wx.Panel(self,-1)
        #定义字符串列表
        langList=['Java','ASP.NET','Python','Ruby','Flex','MVC']
        #生成个数与 langList 列表长度相同的单选按钮，横向排列
        wx.RadioBox(panel,-1,'你最喜欢的一种语言?
',(10,10),(300,100),langList,3,wx.RA_SPECIFY_COLS)
if __name__=='__main__':
    app=wx.PySimpleApp()
    radioBoxFrame=RadioBoxFrame()
    radioBoxFrame.Show()
    app.MainLoop()
```

该段代码首先定义了列表 langList，其中存放了 6 种语言，然后使用 RadioBox 类的构造函数创建一组包含 6 个单选按钮的控件组，并使用 wx.RA_SPECIFY_COLS 将单选按钮控件排列为三列。该段代码运行后的效果如图 14-35 所示。

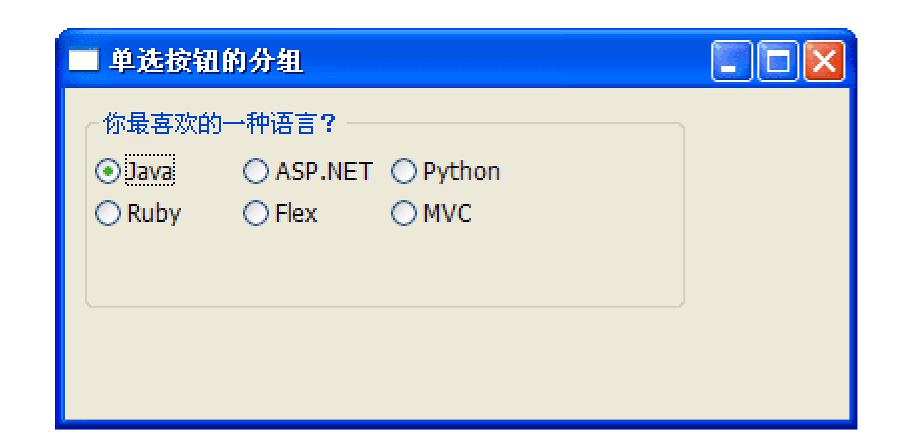


图 14-35　单选按钮的分组

14.4.4　基础知识——多选框

多选框又被称为复选框，是一个带有文本标签的开关按钮。多选框通常成组出现，多选框的状态是布尔类型，用户每次操作可以选择多个多选框。在 wxPython 中，可以使用 wx.CheckBox 类来创建多选框，CheckBox 类的构造函数如下：

```
wx.CheckBox(Window parent, int id, String label = EmptyString, Point pos =
DefaultPosition, Size size = DefaultSize, long style = 0, Validator validator
= DefaultValidator, String name = CheckBoxNameStr)
```

下面使用 CheckBox 类的构造函数来创建一个示例，该示例用于调查用户最喜爱的电影演员，创建 5 个多选框，用户可以选择多个选项。

```
import wx
class CheckBoxFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'多选框',size=(400,200))
        panel=wx.Panel(self,-1)
        wx.StaticText(panel,-1,'你最喜欢的电影演员是:
',pos=(10,10),size=(150,20))
        wx.CheckBox(panel,-1,'陆毅',pos=(10,30),size=(100,20))
        wx.CheckBox(panel,-1,'董洁',pos=(10,50),size=(100,20))
        wx.CheckBox(panel,-1,'姜文',pos=(10,70),size=(100,20))
        wx.CheckBox(panel,-1,'刘亦菲',pos=(10,90),size=(100,20))
        wx.CheckBox(panel,-1,'赵薇',pos=(10,110),size=(100,20))
if __name__=='__main__':
    app=wx.PySimpleApp()
    checkBox=CheckBoxFrame()
    checkBox.Show()
    app.MainLoop()
```

在该段代码中，首先使用 StaticText 类的构造函数创建了一个静态文本框，然后接着使用 CheckBox 类的构造函数创建了 5 个多选框。该段代码运行后的效果如图 14-36 所示。

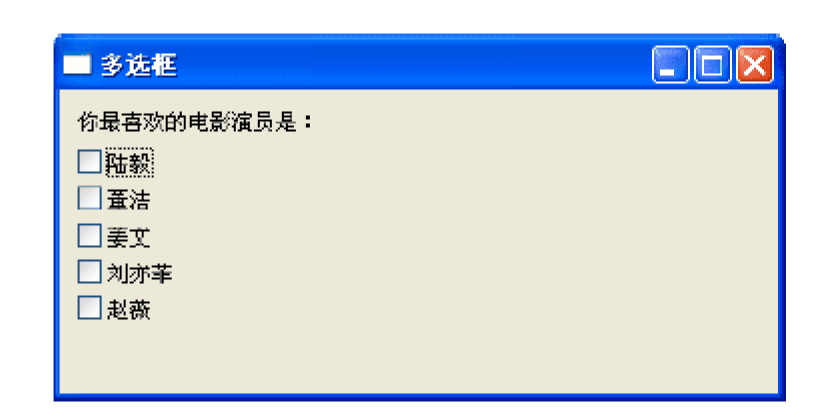


图 14-36　多选框

14.4.5　基础知识——列表控件

列表控件的作用与单选按钮控件和多选框控件相同，都可以为用户提供选择。列表控件的形式多样，包括列表框、下拉列表框等控件。

1. 列表框

列表框把提供给用户选择的选项放到一个矩形的区域中，用户可以在列表框中选择一个或多个选项。ListBox 类可用于创建列表框，ListBox 类的构造函数如下：

```
wx.ListBox(Window parent, int id, Point pos = DefaultPosition, Size size =
DefaultSize, wxArrayString choices = wxPyEmptyStringArray, long style =0,
Validator validator = DefaultValidator, String name = ListBoxNameStr)
```

其中，参数 style 提供了列表框选项的选择方式，style 的样式如表 14-4 所示。

表 14-4　列表框的样式

样　式	描　述
wx.LB_EXTENDED	使用 Shift 键和鼠标连续选择
wx.LB_MULTIPLE	一次可以选择多个选项
wx.LB_SINGLE	一次最多只能选择一个选项
wx.LB_ALWAYS_SB	列表框始终显示一个垂直的滚动条
wx.LB_HSCROLL	列表框始终显示一个水平滚动条
wx.LB_SORT	列表框中的选项按照字母的升序进行排列

下面创建一个示例，具体介绍如何使用 ListBox 类的构造函数创建一个普通列表框。

```
import wx
class ListBoxFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'列表框',pos=(10,10),size=(300,180))
        panel=wx.Panel(self,-1)
        langList=['Java','ASP.NET','Python','Ruby','Flex','MVC']
        self.listBox=wx.ListBox(panel,-1,(10,10),(150,120),langList,
wx.LB_SINGLE) #定义列表框
        self.listBox.SetSelection(0)
        self.Bind(wx.EVT_LISTBOX,self.OnSelected,self.listBox)
    def OnSelected (self,event):
        index=self.listBox.GetSelection()
        wx.MessageBox(self.listBox.GetString(index),'提示')
if __name__=='__main__':
    app=wx.PySimpleApp()
    ListBoxFrame().Show()
    app.MainLoop()
```

在该段代码中，定义了一个长为 150 像素，高为 120 像素的列表框，并调用 SetSelection()

方法，默认情况下选中的是第一个选项。接着使用 Bind()方法绑定选中列表框中的选项后触发的事件。该段代码运行后的效果如图 14-37 所示。当选中一个选项时，弹出选项的文本内容，如图 14-38 所示。

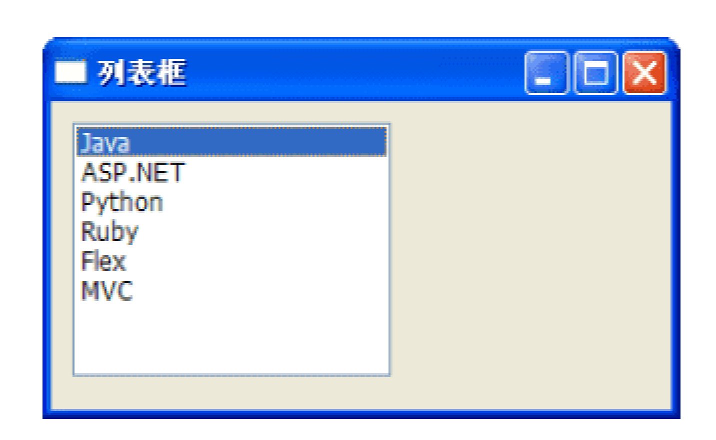


图 14-37　列表框

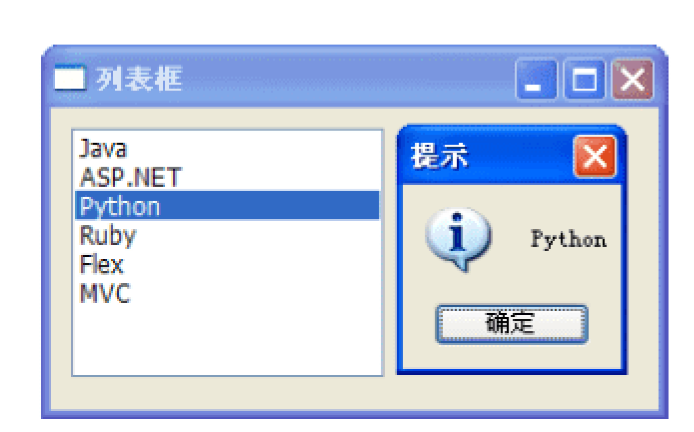


图 14-38　触发选中选项事件

2．带有复选框的列表框

在 wxPython 中，还提供了带有复选框的列表框控件，用于用户选择，丰富了用户界面的友好性。使用 wx.CheckListBox 类可以创建带有复选框的列表框。CheckListBox 类的构造函数如下：

```
wx.CheckListBox(Window parent, int id, Point pos = DefaultPosition, Size size
= DefaultSize, wxArrayString choices = wxPyEmptyStringArray, long style=0,
Validator validator = DefaultValidator, String name = ListBoxNameStr)
```

CheckListBox 类的构造函数中的 style 参数样式与 ListBox 类中的构造函数的 style 参数相同。使用 CheckListBox 类的构造函数创建带有复选框的列表框与使用 ListBox 类的构造函数创建列表框相同。

3．下拉列表框

下拉列表框是列表框的另一种表现形式，下拉列表框占用空间小，适合存放数据量较少的选项。在 wxPython 中，可以使用 Choice 类创建下拉列表框，Choice 类的构造函数如下：

```
wx.Choice(Window parent, int id, Point pos = DefaultPosition, Size size =
DefaultSize, List choices = EmptyList, long style = 0, Validator validator =
DefaultValidator,String name = ChoiceNameStr)
```

下面创建一个示例，具体介绍如何使用 Choice 类创建下拉列表框控件。

```
import wx
class ChoiceFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'下拉列表框',size=(300,200))
        panel=wx.Panel(self,-1)
        langList=['Java','ASP.NET','Python','Ruby','Flex','MVC']
        wx.Choice(panel,-1,pos=(50,10),size=(150,100),choices=langList)
if __name__=='__main__':
    app=wx.PySimpleApp()
    choiceFrame=ChoiceFrame()
    choiceFrame.Show()
    app.MainLoop()
```

运行后的效果如图 14-39 所示。

4．可编辑的下拉列表框

从上面的例子可以看出，使用 Choice 类创建的下拉列表是只读的，用户不能在下拉列表中添加新的选项。如果用户选择的选项不在列表中，应该允许用户自行输入，这才是合理的做法。在 wxPython 中，可以使用 ComboBox 控件来创建一个文本框和一个下拉列表框的组合。ComboBox 类的构造函数如下：

```
wx.ComboBox(Window parent, int id, String value = EmptyString, Point pos =
DefaultPosition, Size size= DefaultSize, List choices = EmptyList, long style
=0,Validator validator = DefaultValidator, String name = ComboBoxNameStr)
```

ComboBox 类的构造函数中的 style 参数提供了列表框选项的显示样式，如表 14-5 所示。

表 14-5　ComboBox控件的样式

样　式	描　述
wx.CB_DROPDOWN	创建一个带有下拉列表的组合框
wx.CB_SIMPLE	创建一个带有列表框的组合框
wx.CB_READONLY	创建一个只读状态的列表框
wx.CB_SORT	创建一个按字母排列的列表框

下面创建一个示例，具体介绍如何使用 ComboBox 控件来选择“喜爱的语言”的方法。

```
import wx
class ChoiceFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'可编辑的下拉列表框',size=(300,200))
        panel=wx.Panel(self,-1)
        langList=['Java','ASP.NET','Python','Ruby','Flex','MVC']
        #定义可编辑的下拉列表框
        wx.ComboBox(panel,-1,'Python',(50,10),(200,150),langList,wx.CB_DROPDOWN)
if __name__=='__main__':
    app=wx.PySimpleApp()
    choiceFrame=ChoiceFrame()
    choiceFrame.Show()
    app.MainLoop()
```

运行后的效果如图 14-40 所示。



图 14-39　下拉列表框



图 14-40　可编辑的下拉列表框



ComboBox 类是 Choice 类的子类，在使用 ComboBox 类创建可编辑的下拉列表框时，可使用 Choice 类中的方法。

14.4.6 基础知识——Sizers布局组件

Sizers 布局管理器是管理界面中各种控件的组件，使用 Sizers 组件可以自动解决控件的位置和控件之间的间距问题，提高了 GUI 程序的可控性。

1. Grid布局管理器

前面的程序都是通过设置构造函数的 pos 和 size 属性调整控件之间的位置，wxPython 中的 Grid 布局管理器采用表格的形式分配各种控件，不需要设置控件在容器中的位置，直接添加到 Grid Sizer 布局管理器中即可。使用 wx.GridSizer 类可以创建一个 Grid 布局管理器，该管理器是一个固定的二维网格，其中的每个元素都有相同的尺寸。当创建一个 Grid Sizer 时，需要固定行的数量或者列的数量，元素以从左到右添加，直到一行被填满，然后从下一行开始。GridSizer 类的构造函数格式如下：

```
wx.GridSizer(int rows=0,int cols=0,int hgap=0,int vgap=0)
```

GridSizer 类的构造函数有 4 个参数，分别指定布局表格的行列数目以及组件之间的水平距离和垂直距离。

创建布局管理器需要用到 Add()方法，该方法用于将各个组件添加到布局管理器中，Add()方法的声明如下：

```
Add(Window window, integer proportion=0, integer flag = 0, integer border = 0)
```

其中：

- window：要添加到布局管理器的组件。
- proportion：当窗体的大小发生变化时，控件之间的比例。假设将 3 个按钮添加到一个布局管理器中，它们的 proportion 参数的值分别为 0、1 和 2，则当窗口发生改变时，proportion 参数被设置为 0 的按钮不会在水平方向上发生改变，而被设置为 2 的按钮的大小始终是设置为 1 的按钮的两倍。
- flag：指定布局管理器的边框。
- border：边框的大小，可以使用 wx.LEFT、wx.RIGHT、wx.BOTTOM、wx.TOP 和 wx.ALL 作为该参数的值，也可以用这些值的组合作为 border 参数的值。

下面使用 wx.GridSizer 类的构造函数来构造一个计算器的基本骨架。

```
import wx
class GridSizer(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(300, 200))
        panel=wx.Panel(self,-1)
        gs = wx.GridSizer(4, 4, 15,15)  #创建 4 行 4 列的布局，水平间距和垂直间距都为 15
        numList=['Cls','Bck','','Close','7','8','9','/','4','5','6','*','1',
'2','3','-','0','.','=','+']
        for num in numList :         #循环遍历集合，创建多个按钮控件
            btn=wx.Button(panel,id,num)
            gs.Add(btn,0,0)       #将按钮组件添加到布局管理器中
        panel.SetSizer(gs)       #将 GridSizer 管理器添加到容器中
        self.Centre()
```

```
        self.Show(True)         #显示窗口
app = wx.App()
GridSizer(None, -1, 'GridSizer')
app.MainLoop()
```

在以上代码中，首先创建了一个 GridSizer 布局管理器，整个布局为 4 行 4 列，组件之间的水平距离和垂直距离均为 15，然后通过循环遍历 16 个 Button 控件，并调用管理器的 Add()方法将各个按钮控件添加到布局管理器中，接着调用 SetSizer()方法将布局管理器添加到容器中。最后使用 Show()方法显示窗口，因此在后面调用 GridSizer 类时不需要再次使用 Show()方法显示窗口。运行该段代码，效果如图 14-41 所示。

2. Flex Grid布局管理器

Flex Grid 布局管理器与 Grid 布局管理器相似，但提供了更多的灵活性。在 Grid 布局中，所有的单元格都必须大小相同，但在 Flex Grid 布局中，同行的单元格的高度相同，同列的单元格的宽度相同，而不同的列和行可以高度不同。在 wxPython 中，可以使用 wx.FlexGridSizer 类创建 Flex Grid 局管理器，FlexGridSizer 类的构造函数如下：

```
FlexGridSizer(int rows=1, int cols=0, int vgap=0, int hgap=0)
```

下面创建一个示例，具体介绍如何使用 wx.FlexGridSizer 类的构造函数来创建 Flex Grid 布局管理器。

```
import wx
class FlexGridSizer(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(300, 200))
        panel = wx.Panel(self, -1)
        fgs = wx.FlexGridSizer(3, 2, 9, 25)      #定义 Flex Grid Sizer 布局管理器
        title = wx.StaticText(panel, -1, '标题') #定义一个 titile 静态文本框
        tc1 = wx.TextCtrl(panel, -1)             #定义 tc1 输入文本框
        fgs.Add(title,0,0)          #将 title 文本框添加到 Flex Grid Sizer 布局管理器
        fgs.Add(tc1,0,0)            #将 tc1 文本框添加到 Flex Grid Sizer 布局管理器
        author = wx.StaticText(panel, -1, '作者')
        tc2 = wx.TextCtrl(panel, -1)
        fgs.Add(author,0,0)         #将 author 文本框添加到 Flex Grid Sizer 布局管理器
        fgs.Add(tc2,0,0)            #将 tc2 文本框添加到 Flex Grid Sizer 布局管理器
        review = wx.StaticText(panel, -1, '内容',size=(-1,100))
        tc3 = wx.TextCtrl(panel, -1,size=(-1,100),style=wx.TE_MULTILINE)
        fgs.Add(review,0,0)         #将 review 文本框添加到 Flex Grid Sizer 布局管理器
        fgs.Add(tc3,0,0)            #将 tc3 文本框添加到 Flex Grid Sizer 布局管理器
        panel.SetSizer(fgs)         #将 Flex Grid Sizer 布局管理器添加到 panel 中
        self.Centre()
        self.Show(True)
app = wx.App()
FlexGridSizer(None, -1, 'FlexGridSizer')
app.MainLoop()
```

在该段代码中，首先使用 wx.FlexGridSizer 类的构造函数创建了一个 3 行 2 列的布局，然后将 6 个文本框控件添加到 Flex Grid 布局管理器中，并使用容器的 SetSizer()方法将已经创建的 Flex Grid Sizer 布局管理器添加到容器中。运行该段代码，效果如图 14-42 所示。

图 14-41　Grid布局管理器



图 14-42　Flex Grid布局管理器

3. Grid Bag布局管理器

Grid Bag 布局提供了更强大的功能。它也是 wxPython 中最复杂的一个布局，它可以精确地定位组件。下面是 wx.GridBagSizer 的构造函数。

```
wx.GridBagSizer(integer vgap, integer hgap)
```

从构造函数的参数来看，并不需要设置布局的行和列，只需要设置间距即可，这也正是它的优势所在。下面创建一个示例，体验一下该布局的魅力吧。

```
import wx
class OpenResource(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(400, 400))
        panel = wx.Panel(self, -1)
        sizer = wx.GridBagSizer(4, 4)              #创建 Grid Bag 布局管理器
        #创建静态文本框
        text1 = wx.StaticText(panel, -1, 'Select a resource to open')
        #将静态文本框添加到布局管理器中
        sizer.Add(text1, (0, 0), flag=wx.TOP | wx.LEFT | wx.BOTTOM, border=5)
        tc = wx.TextCtrl(panel, -1)                #创建一个输入文本框
        sizer.Add(tc, (1, 0), (1, 3), wx.EXPAND | wx.LEFT | wx.RIGHT, 5)
        text2 = wx.StaticText(panel, -1, 'Matching resources')
        sizer.Add(text2, (2, 0), flag=wx.TOP | wx.LEFT | wx.BOTTOM, border=5)
        list1 = wx.ListBox(panel, -1, style=wx.LB_ALWAYS_SB)  #创建一个空的列表框
        sizer.Add(list1, (3, 0), (5, 3), wx.EXPAND | wx.LEFT | wx.RIGHT, 5)
        text3 = wx.StaticText(panel, -1, 'In Folders')
        sizer.Add(text3, (8, 0), flag=wx.TOP | wx.LEFT | wx.BOTTOM, border=5)
        list2 = wx.ListBox(panel, -1, style=wx.LB_ALWAYS_SB)
        sizer.Add(list2, (9, 0), (3, 3), wx.EXPAND | wx.LEFT | wx.RIGHT, 5)
        cb = wx.CheckBox(panel, -1, 'Show derived resources')    #创建一个多选框
        #将多选框添加到布局管理器中
        sizer.Add(cb, (12, 0), flag=wx.LEFT | wx.RIGHT, border=5)
        buttonOk = wx.Button(panel, -1, 'OK', size=(90, 28))      #创建一个按钮
        buttonCancel = wx.Button(panel, -1, 'Cancel', size=(90, 28))
        sizer.Add(buttonOk, (14, 1))        #将上面创建的两个按钮添加到布局管理器中
        sizer.Add(buttonCancel, (14, 2), flag=wx.RIGHT | wx.BOTTOM, border=5)
        #创建一个位图按钮
        help = wx.BitmapButton(panel, -1, wx.Bitmap('ico/15b53eda7255d444d1164e2e.png'),
 style=wx.NO_BORDER)
        #将位图按钮添加到布局管理器中
        sizer.Add(help, (14, 0), flag=wx.LEFT, border=5)
        panel.SetSizer(sizer)           #将布局管理器添加到 panel 中
        self.Centre()
        self.Show(True)
app = wx.App()
```

```
OpenResource(None, -1, 'Open Resource')
app.MainLoop()
```

在该段代码中，首先创建了一个 FlexGridSizer 类型的布局，整个布局分为 8 行，组件之间的水平间距和垂直间距都为 4。接着将静态文本框、输入文本框、列表框、多选框、按钮和位图按钮进行了自由布局。该段代码运行的结果如图 14-43 所示。

4. Box布局管理器

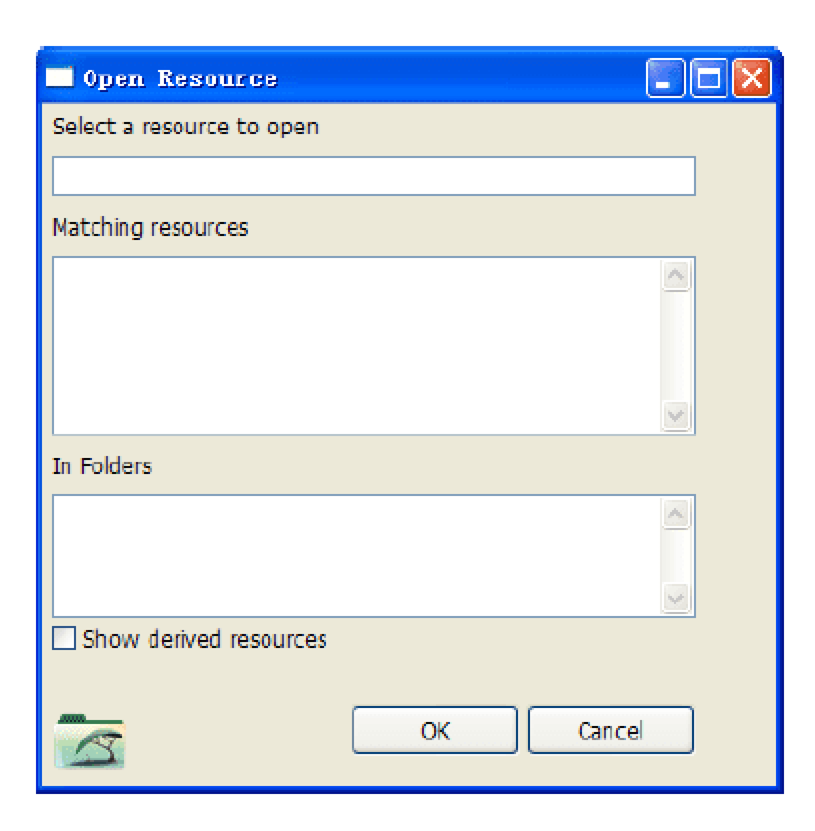
Box Sizer 是布局管理器中最简单、最灵活的一种布局。它可以使放置其上的组件成行或者成列排列，使各个组件从左至右或从上到下排列在一条线上。还可以放置一种布局到另一布局中，嵌套使用，以便创建复杂的布局。使用 wx.BoxSizer 类可以创建 Box 布局管理器，BoxSizer 类的构造函数如下：

```
wx.BoxSizer(integer orient)
```

参数 orient 表示排列的方式，该参数的值有两个：wx.VERTICAL 和 wx.HORIZONTAL。下面创建一个示例，具体介绍如何使用 BoxSizer 类的构造函数来创建 Box 布局管理器。

```
import wx
class Border(wx.Frame):
    def __init__(self, parent, id, title):
        wx.Frame.__init__(self, parent, id, title, size=(250, 200))
        panel = wx.Panel(self, -1)
        panel.SetBackgroundColour('white')
        vbox = wx.BoxSizer(wx.VERTICAL)      #创建 Box 布局管理器
        langList=['Java','ASP.NET','Python','Ruby','Flex','MVC']
        for lang in langList :
            btn=wx.Button(panel,id,lang)
            vbox.Add(btn,0,wx.EXPAND | wx.ALL,2)      #将按钮控件添加到布局管理器中
        panel.SetSizer(vbox)                          #将布局管理器添加到容器中
        self.Centre()
        self.Show(True)
app = wx.App()
Border(None, -1,'')
app.MainLoop()
```

该段代码首先创建一个垂直排列的布局 BoxSizer，并使用 Add()方法将按钮控件添加到布局管理器中，然后使用 SetSizer()方法将布局管理器添加到容器中。该段代码运行后的效果如图 14-44 所示。



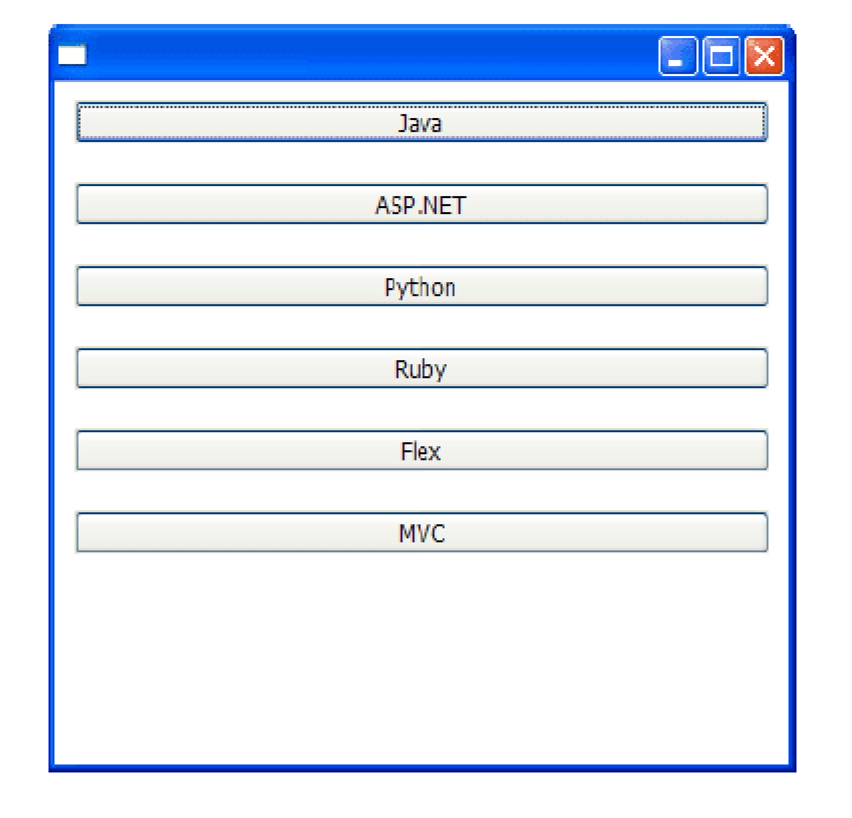


图 14-43　Grid Bag布局管理器　　　图 14-44　Box布局管理器

14.4.7　实例描述

用过 MyEclipse 开发工具的程序员应该知道，当按 Shift+F 键时，会弹出“查找/替换”窗口。在该窗口中，可以输入要查找的字符和将要替换的字符，还可以选择从哪个位置开始查找。下面使用 wxPython 的一些基本组件和布局管理器来创建一个类似于 MyEclipse 开发工具中的“查找/替换”窗口。

14.4.8　实例应用

【例 14-3】使用 wxPython 的基本组件和布局管理器创建 Find/Replace 窗口。

(1) 新建 Python 文件，命名为 wx_3.py。

(2) 在 wx_3.py 文件中编辑代码。首先将 wx 包导入该文件，并创建 FindReplace 类，然后在该类的__init__()方法中创建两个 Box 布局管理器(vbox_top 和 vbox)和一个面板(panel)，代码如下：

```
import wx
class FindReplace(wx.Frame):
   def __init__(self, parent, id, title):
      wx.Frame.__init__(self, parent, id,title, size=(255, 365))
      vbox_top = wx.BoxSizer(wx.HORIZONTAL)
      panel = wx.Panel(self, -1)
      vbox = wx.BoxSizer(wx.VERTICAL)
```

(3) 继续在__init__()方法中编辑代码。创建第一个面板及面板上的组件。使用 wx.GridSizer 类的构造函数创建一个 2 行 2 列的 Grid 布局管理器，并使用 Add()方法将两个静态文本输入框和两个下拉列表控件添加到 Grid 布局管理器中，最后将该 Grid 布局管理器添加到面板中，同时将该面板添加到 vbox 布局管理器中。代码如下：

```
panel1 = wx.Panel(panel, -1)
      grid1 = wx.GridSizer(2, 2)
      grid1.Add(wx.StaticText(panel1, -1, 'Find: ', (5, 5)), 0,
wx.ALIGN_CENTER_VERTICAL)
      grid1.Add(wx.ComboBox(panel1, -1, size=(120, -1)))
      grid1.Add(wx.StaticText(panel1, -1, 'Replace with: ', (5, 5)), 0,
wx.ALIGN_CENTER_VERTICAL)
      grid1.Add(wx.ComboBox(panel1, -1, size=(120, -1)))
      panel1.SetSizer(grid1)
      vbox.Add(panel1, 0, wx.BOTTOM | wx.TOP, 9)
```

(4) 创建第二个面板，并创建横向排列的 Box 布局管理器。该布局为一行两列的布局。每一列都由 StaticBox 布局管理器来控制。代码如下：

```
      panel2 = wx.Panel(panel, -1)
      hbox2 = wx.BoxSizer(wx.HORIZONTAL)
      sizer21 = wx.StaticBoxSizer(wx.StaticBox(panel2, -1, 'Direction'),
orient=wx.VERTICAL)
      sizer21.Add(wx.RadioButton(panel2, -1, 'Forward', style=wx.RB_GROUP))
```

```
        sizer21.Add(wx.RadioButton(panel2, -1, 'Backward'))
        hbox2.Add(sizer21, 1, wx.RIGHT, 5)
        sizer22 = wx.StaticBoxSizer(wx.StaticBox(panel2, -1, 'Scope'),
orient=wx.VERTICAL)
        sizer22.Add(wx.RadioButton(panel2, -1, 'All', style=wx.RB_GROUP))
        sizer22.Add(wx.RadioButton(panel2, -1, 'Selected Lines'))
        hbox2.Add(sizer22, 1)
        panel2.SetSizer(hbox2)
        vbox.Add(panel2, 0, wx.BOTTOM, 9)
```

(5) 创建第三个面板，该面板由 StaticBox 与 Box 布局的嵌套使用来控制组件的分布。代码如下：

```
        panel3 = wx.Panel(panel, -1)
        sizer3 = wx.StaticBoxSizer(wx.StaticBox(panel3, -1, 'Options'),
orient=wx.VERTICAL)
        vbox3 = wx.BoxSizer(wx.VERTICAL)
        grid = wx.GridSizer(3, 2, 0, 5)
        grid.Add(wx.CheckBox(panel3, -1, 'Case Sensitive'))
        grid.Add(wx.CheckBox(panel3, -1, 'Wrap Search'))
        grid.Add(wx.CheckBox(panel3, -1, 'Whole Word'))
        grid.Add(wx.CheckBox(panel3, -1, 'Incremental'))
        vbox3.Add(grid)
        vbox3.Add(wx.CheckBox(panel3, -1, 'Regular expressions'))
        sizer3.Add(vbox3, 0, wx.TOP, 4)
        panel3.SetSizer(sizer3)
        vbox.Add(panel3, 0, wx.BOTTOM, 15)
```

(6) 编辑第四个面板，使用 2 行 2 列的 Grid 布局管理器来控制 4 个按钮的排列，代码如下：

```
        panel4 = wx.Panel(panel, -1)
        sizer4 = wx.GridSizer(2, 2, 2, 2)
        sizer4.Add(wx.Button(panel4, -1, 'Find', size=(120, -1)))
        sizer4.Add(wx.Button(panel4, -1, 'Replace/Find', size=(120, -1)))
        sizer4.Add(wx.Button(panel4, -1, 'Replace', size=(120, -1)))
        sizer4.Add(wx.Button(panel4, -1, 'Replace All', size=(120, -1)))
        panel4.SetSizer(sizer4)
        vbox.Add(panel4, 0, wx.BOTTOM, 9)
```

(7) 编辑第五个面板，代码如下：

```
        panel5 = wx.Panel(panel, -1)
        sizer5 = wx.BoxSizer(wx.HORIZONTAL)
        sizer5.Add((191, -1), 1, wx.EXPAND | wx.ALIGN_RIGHT)
        sizer5.Add(wx.Button(panel5, -1, 'Close', size=(50, -1)))
        panel5.SetSizer(sizer5)
        vbox.Add(panel5, 1, wx.BOTTOM, 9)
```

(8) 将含有 5 个面板的 Box 布局管理器(vbox)添加到 vbox_top 布局管理器中，并将该布局管理器添加到 Panel 中，最后使用 Show()方法显示窗口。代码如下：

```
        vbox_top.Add(vbox, 1, wx.LEFT, 5)
        panel.SetSizer(vbox_top)
        self.Centre()
        self.Show()
```

(9) 调用 FindReplace 类，显示窗口。代码如下：

```
app = wx.App()
FindReplace(None, -1, 'Find/Replace')
app.MainLoop()
```

14.4.9 运行结果

运行 wx_3.py 文件，运行效果如图 14-45 所示。

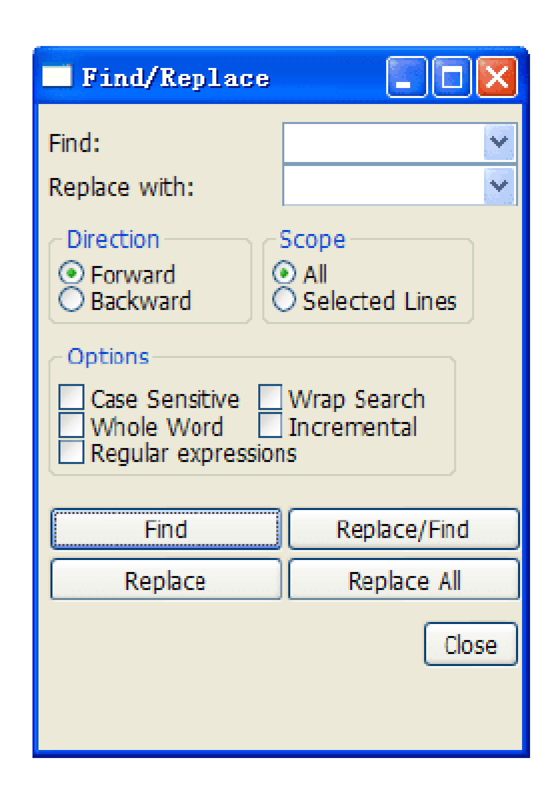


图 14-45　Find/Replace窗口

14.4.10　实例分析



源码解析

该实例首先将 5 个面板(Panel)添加到 Box 布局管理器中，该布局管理器的排列模式为 VERTICAL，然后再将包含有 5 个面板的 Box 布局管理器添加到一个排列模式为 HORIZONTAL 的 Box 布局管理器中，并将该 Box 布局管理器添加到面板中。

14.5　wxPython库中的菜单控件

菜单是系统的功能列表，其形式灵活多样，包括多级菜单、位图菜单以及上下文菜单等。本节将详细介绍 wxPython 库中的菜单控件。



视频教学：光盘/videos/14/菜单的创建和使用.avi　　长度：11 分钟

14.5.1　基础知识——菜单的创建和使用

菜单是桌面应用程序中最常用的控件之一，菜单展现了应用程序包含的各种功能。本节将介绍菜单的创建、事件、位图菜单和上下文菜单等方面的内容。

1．创建菜单

菜单的创建一般由以下步骤来完成。

(1) 调用 wx.MenuBar 类创建菜单栏。

(2) 调用 wx.Menu 类创建父菜单。

(3) 调用 MenuBar 类的 Append()方法将父菜单添加到菜单栏中。例如：

```
menuBar.Append(menu, '文件')
```

其中，menuBar 是 MenuBar 类的对象，menu 是 Menu 类的对象。该行代码表示添加了一个“文件”菜单。MenuBar 类的 Append()方法的声明如下：

```
Append(Menu menu, String title)
```

(4) 调用 Menu 类的 Append()方法添加子菜单，代码如下：

```
menu.Append(1, '文件')
```

该行代码表示向 menu 菜单中添加了“文件”子菜单，1 表示子菜单的编号，菜单的编号是唯一的。

(5) 添加菜单事件，菜单事件为 wx.EVT_MENU。



大家可以按照上面的步骤创建一个简单的菜单。

2. 菜单事件

菜单事件是对菜单处理的响应方式，菜单事件包括单击、选择等操作。菜单事件如表 14-6 所示。

表 14-6　菜单的响应事件

事　件	描　述
EVT_MENU	选择菜单时触发的事件
EVT_MENU_CLOSE	关闭菜单时触发的事件
EVT_MENU_HIGHLIGHT	菜单高亮显示时触发的事件
EVT_MENU_HIGHLIGHT_ALL	任何菜单高亮显示时都执行同一个操作
EVT_MENU_OPEN	打开菜单时触发的事件

每个菜单事件的响应都对应一个事件处理函数，当遇到特定的操作时，将触发相应的事件，然后调用事件处理函数处理。下面做一个示例，创建一个 File 父菜单，并创建它的子菜单 About 和 Exit。当选择 About 菜单时，弹出提示对话框；当选择 Exit 菜单时，关闭程序。代码如下：

```
import wx
ID_ABOUT = 101
ID_EXIT  = 102
class MyFrame(wx.Frame):
    def __init__(self, parent, ID, title):
        wx.Frame.__init__(self, parent, ID, title, wx.DefaultPosition,
wx.Size(400, 150))
        self.CreateStatusBar()
        self.SetStatusText("汇智科技有限公司所有权")
        menuBar = wx.MenuBar()            #创建菜单栏
        menu = wx.Menu()                  #创建菜单
        menuBar.Append(menu, "File");     #将 File 父菜单添加到菜单栏中
        #向菜单中添加子菜单
        menu.Append(ID_ABOUT, "About", "More information about this program")
        menu.AppendSeparator()            #添加分割线
        #向菜单中添加子菜单
        menu.Append(ID_EXIT, "E&xit", "Terminate the program")
        self.SetMenuBar(menuBar)          #将父菜单添加到窗口中
        #绑定事件
        wx.EVT_MENU(self, ID_ABOUT, self.OnAbout)
        wx.EVT_MENU(self, ID_EXIT,  self.TimeToQuit)
    #当选择 About 菜单时，弹出提示对话框
    def OnAbout(self, event):
        dlg = wx.MessageDialog(self, "This sample program shows off\n"
                      "frames, menus, statusbars, and this\n"
                      "message dialog.",
                      "About Me", wx.OK | wx.ICON_INFORMATION)
```

```
        dlg.ShowModal()          #以模式窗口打开
        dlg.Destroy()
    #选择 Exit 菜单时，关闭程序
    def TimeToQuit(self, event):
        self.Close(True)
class MyApp(wx.App):
    def OnInit(self):
        frame = MyFrame(None, -1, "Hello from wxPython")
        frame.Show(True)
        self.SetTopWindow(frame)
        return True
app = MyApp(0)
app.MainLoop()
```

运行效果如图 14-46 所示。当选择 File 菜单时，弹出 About 和 Exit 两个子菜单，当选择 About 菜单时，弹出对话框，如图 14-47 所示；当选择 Exit 菜单时，程序关闭。

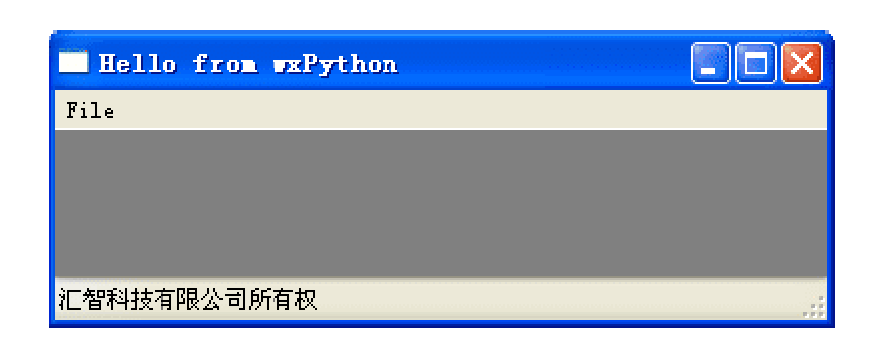


图 14-46　含有菜单栏的窗口



图 14-47　触发About菜单事件

3. 位图菜单

传统菜单的展现方式比较单一，位图菜单很好地解决了这个问题。位图菜单可以在菜单项的左侧添加 BMP 和 PNG 格式的图片，丰富了应用程序的界面。位图菜单的创建和普通菜单的创建过程相同，菜单项使用 MenuItem 对象创建，只是在添加菜单项前需要指定 Bitmap 对象，最后把 Bitmap 对象添加到 MenuItem 中。

下面创建一个示例，具体介绍如何创建一个位图菜单。

```
import wx
class MyFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self,None,-1,"Fancier Menu Example")
        p = wx.Panel(self)
        menu = wx.Menu()
        bmp = wx.Bitmap("ico/file.png", wx.BITMAP_TYPE_PNG)
        fileOpen = wx.MenuItem(menu, -1, "Open")
        fileOpen.SetBitmap(bmp)     #增加一个自定义的位图
        menu.AppendItem(fileOpen)
        font = wx.SystemSettings.GetFont(wx.SYS_DEFAULT_GUI_FONT)
        font.SetWeight(wx.BOLD)
        item = wx.MenuItem(menu, -1, "Has Bold Font")
        item.SetFont(font)#改变字体
        menu.AppendItem(item)
        exitFrame = menu.Append(-1, "Exit")
        self.Bind(wx.EVT_MENU, self.OnExit, exitFrame)
        menuBar = wx.MenuBar()
```

```
        menuBar.Append(menu, "Menu")
        self.SetMenuBar(menuBar)
    def OnExit(self, event):
        self.Close()
#省略调用 MyFrame 类的代码
```

在该段代码中，使用 wx.Bitmap()类的构造函数创建了一个 Bitmap 对象，接着将该对象添加到 Open 菜单项(MenuItem)中，并将 Open 菜单项添加到菜单中(Menu)。运行该段代码后的效果如图 14-48 所示。

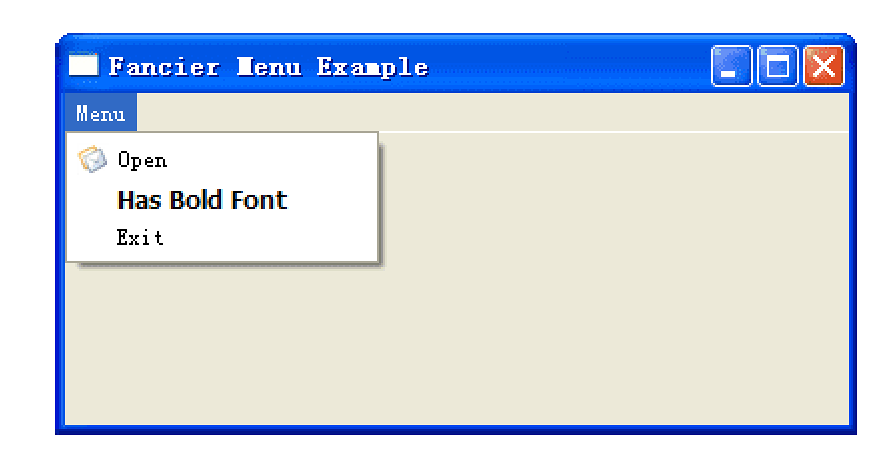


图 14-48　位图菜单

4．上下文菜单

上下文菜单又称弹出式菜单，即单击鼠标右键后弹出的菜单。上下文菜单简化了桌面应用程序的操作，用户可以快速定位到上下文菜单中指定的某个功能。上下文菜单也是由 Menu 类创建的，只要在上下文显示的容器中绑定 wx.EVT_CONTEXT_MENU 事件，然后由容器组件调用 PopupMenu()方法弹出上下文菜单即可。下面创建一个示例，介绍如何创建上下文菜单。

```
import wx
class ContextMenuFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'上下文菜单',size=(400,200))
        self.panel=wx.Panel(self)
        #创建静态文本框
        self.staticText=wx.StaticText(self.panel,-1,'语言查询: ',pos=(10,30))
        #创建输入文本框
        self.inputCtrl=wx.TextCtrl(self.panel,-1,pos=(80,30),size=(200,-1))
        self.popupmenu=wx.Menu()                                #创建菜单
        langList=['Python','Java','ASP.NET','ExtJS']           #创建列表
        #循环列表，将列表中的元素添加至菜单中作为菜单项
        for menu in langList :
            lang=self.popupmenu.Append(-1,menu)
            #绑定菜单项的选中事件
            self.Bind(wx.EVT_MENU,self.OnMenuItemSelected,lang)
            #绑定输入文本框的上下文菜单事件
            self.inputCtrl.Bind(wx.EVT_CONTEXT_MENU, self.OnPopup)
    def OnMenuItemSelected (self,event):
        #通过事件触发的 Id 值获取选中的菜单项
        item=self.popupmenu.FindItemById(event.GetId())
        #将菜单项的内容写入到输入文本框中
        self.inputCtrl.SetLabel(item.GetText())
    def  OnPopup(self,event):
        pos=self.panel.ScreenToClient(event.GetPosition())    #获取鼠标位置
        self.panel.PopupMenu(self.popupmenu,pos)    #在鼠标所在的位置绑定上下文菜单
if __name__=='__main__':
```

```
    app=wx.PySimpleApp()
    frame=ContextMenuFrame()
    frame.Show()
    app.MainLoop()
```

在该段代码中，将一个列表中的元素作为菜单项，并绑定了每个菜单项的选择事件，同时绑定了输入文本框的上下文菜单事件，然后分别创建了这两个事件所对应的方法。运行该段代码，显示一个静态文本框和一个输入文本框，右击输入文本框，出现上下文菜单，如图 14-49 所示。选择其中的一项，文本框中则显示选择项的内容，如图 14-50 所示。

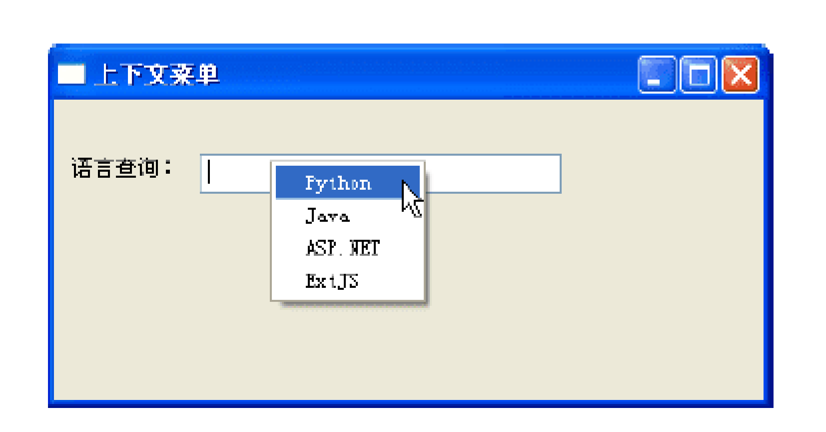


图 14-49　上下文菜单

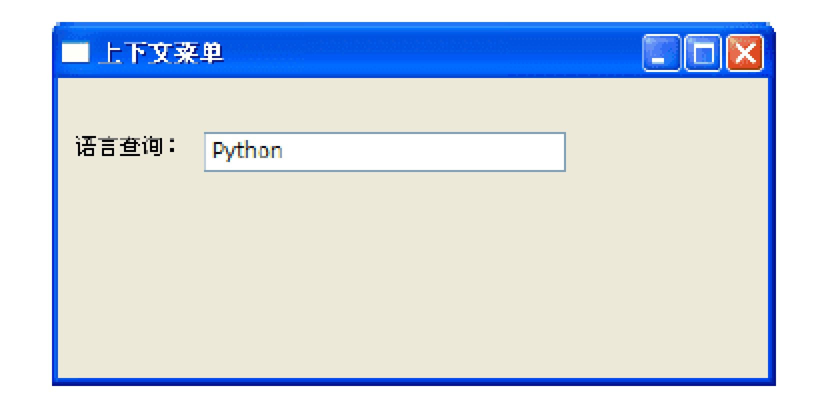


图 14-50　输入文本框中显示菜单项的内容

14.5.2　实例描述

当打开任意一个开发工具时，会发现菜单栏中有一排主菜单，每个主菜单并不是只有一级菜单，还拥有二级菜单，甚至更多，并且很多菜单项又可以使用快捷键来进行操作。当在开发工具中刚打开一个文件时，该文件中的“撤销”菜单是禁用的，当编辑一段代码之后，“撤销”菜单才会被启用，诸如此类的菜单功能在开发工具中很重要，这样不仅给用户提供了方便，同时也避免了误操作。下面创建一个案例，实现菜单的快捷键、多级菜单和高亮显示等功能。

14.5.3　实例应用

【例 14-4】 制作 EditPlus 的菜单项。

(1) 新建 wxPython 文件，命名为 wx_4.py。

(2) 创建 MainFrame 类，在该类的初始化方法中，创建一个窗口，并在该窗口的面板中创建 5 个主菜单，分别为文件、编辑、视图、系统和设置。“文件”菜单是一个多级菜单，其他的主菜单都只有一级菜单，同时为“文件”主菜单下的“打开”子菜单添加快捷键。MainFrame 类的__init__()方法的代码如下：

```
import wx
import os
class MainFrame ( wx.Frame ):
    def __init__( self, parent ):
        wx.Frame.__init__(self,parent,-1,'EditPlus',size=(500,300))
        panel=wx.Panel(self)
        menuBar=wx.MenuBar()        #创建菜单栏
        fileMenu=wx.Menu()          #创建菜单对象
        sFileMenu=wx.Menu()                        #创建二级菜单
        sFileMenu.Append(11,'标准文本')
        sFileMenu.Append(12,'HTML 网页')
        fileMenu.AppendMenu(-1,'新建(N)',sFileMenu)#向主菜单的“新建”子菜单添加菜单项
```

```
        fileMenu.Append(2,'&o 打开(O)')         #向主菜单添加“打开”子菜单，并使用快捷键 o
        wx.EVT_MENU(self, 2, self.OnOpen)       #绑定“打开”菜单事件
        fileMenu.Append(3,'关闭(C)')            #向主菜单添加“关闭”子菜单
        menuBar.Append(fileMenu, "文件");       #将主菜单添加到菜单栏中
        fileMenu.AppendSeparator()              #添加分割线
        editMenu=wx.Menu()
        editMenu.Append(4,'&Z 撤销')            #创建快捷键“Z”
        editMenu.Append(5,'&Y 重做')
        editMenu.Append(6,'剪切')
        editMenu.Append(7,'重做')
        menuBar.Append(editMenu,'编辑')
        editMenu.AppendSeparator()
        viewMenu=wx.Menu()
        menuBar.Append(viewMenu,'视图')
        self.exitMenu=wx.Menu()                 #创建菜单对象
        self.exitMenu.Append(1000,'退出')       #向菜单对象中添加“退出”子菜单
        menuBar.Append(self.exitMenu,'系统菜单')  #将“退出”子菜单添加到“系统菜单”中
        sysMenu=wx.Menu()
        subSetMenu=sysMenu.Append(1001,'打开/屏蔽菜单')    #定义 id 为 1001 的菜单项
        self.Bind(wx.EVT_MENU,self.OnExit,id=1000)
        #绑定 id 为 1000 的菜单，当高亮显示时，触发事件
        self.Bind(wx.EVT_MENU_HIGHLIGHT,self.OnItemSelected,id=1000)
        #绑定“打开/屏蔽”菜单的菜单项
        self.Bind(wx.EVT_MENU,self.OnEnable,subSetMenu)
        menuBar.Append(sysMenu,'设置')
        self.SetMenuBar(menuBar)                #将父菜单添加到窗口中
        self.Show()
```

(3) 当单击“打开”菜单或者按快捷键 o 时，触发 EVT_MENU 事件，即执行 OnOpen()方法，打开一个特殊对话框，选择要打开的文件。继续在 mainFrame 类中编辑 OnOpen()方法，代码如下：

```
def OnOpen (self,event):
        filterFile='Python Source(*.py)|*.py|All files(*.*)|*.*'
        dialog=wx.FileDialog(None,'选择文件',os.getcwd(),'',filterFile,wx.OPEN)
        dialog.ShowModal()
        dialog.Destroy()
```

(4) 当选择“打开/屏蔽”菜单项时，触发该菜单项的 EVT_MENU 事件，即执行 OnEnable()方法，在该方法中获取 Id 为 1000 的菜单项是否为禁用状态，如果非禁用状态，则设置为禁用，否则设置为启用状态。OnEnable()方法的代码如下：

```
def OnEnable (self,event):
        menuBar=self.GetMenuBar()
        enabled=menuBar.IsEnabled(1000)         #获取“退出”菜单是否为禁用状态
        self.exitMenu.Enable(1000,not enabled)
```

(5) 当选择“退出”菜单时，关闭窗口，代码如下：

```
def OnExit (self,event):
        self.Close()
```

(6) 当“退出”菜单项(Id 为 1000 的菜单项)高亮显示时，触发 EVT_MENU_HIGHLIGHT 事件，即执行 OnItemSelected()方法。在该方法中，使用 FindItemById()方法获取被选择的菜单

项，并弹出该菜单项的内容。OnItemSelected()方法的定义如下：

```
def OnItemSelected (self,event):
        item=self.GetMenuBar().FindItemById(event.GetId())
        wx.MessageBox('Menu:'+item.GetText())
```

(7) 调用 MainFrame 类，显示窗口，代码如下：

```
app=wx.PySimpleApp()
MainFrame(None).Show()
app.MainLoop()
```

14.5.4　运行结果

运行 wx_4.py 文件，出现图 14-51 所示的窗口。当选择“文件”主菜单时，弹出包含三项的一级菜单，将鼠标放到一级菜单的“新建”菜单项上，弹出包含有两个菜单项的二级菜单，如图 14-52 所示。当选择一级菜单中的“打开”子菜单时(或将鼠标放到该菜单上同时按下 o 键)，弹出“选择文件”对话框，如图 14-53 所示。

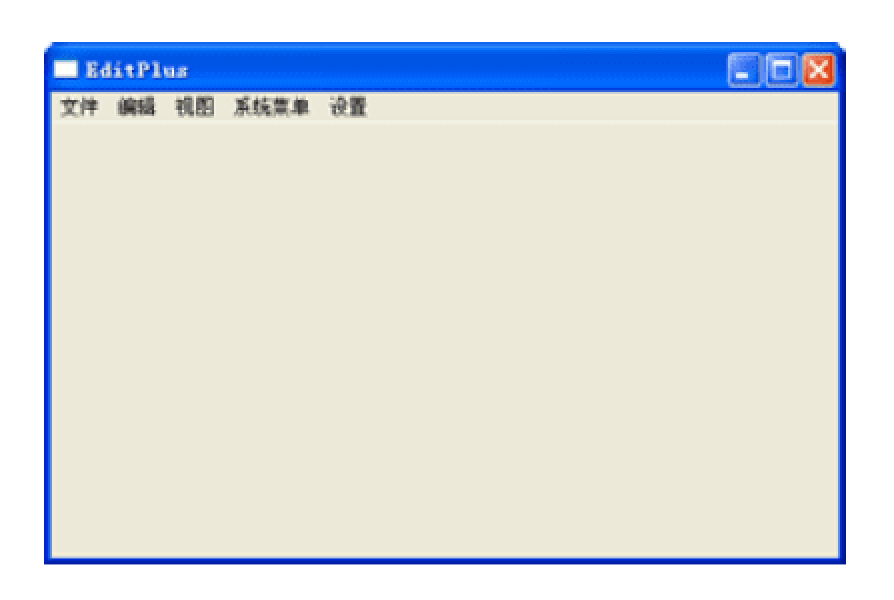


图 14-51　EditPlus的主菜单

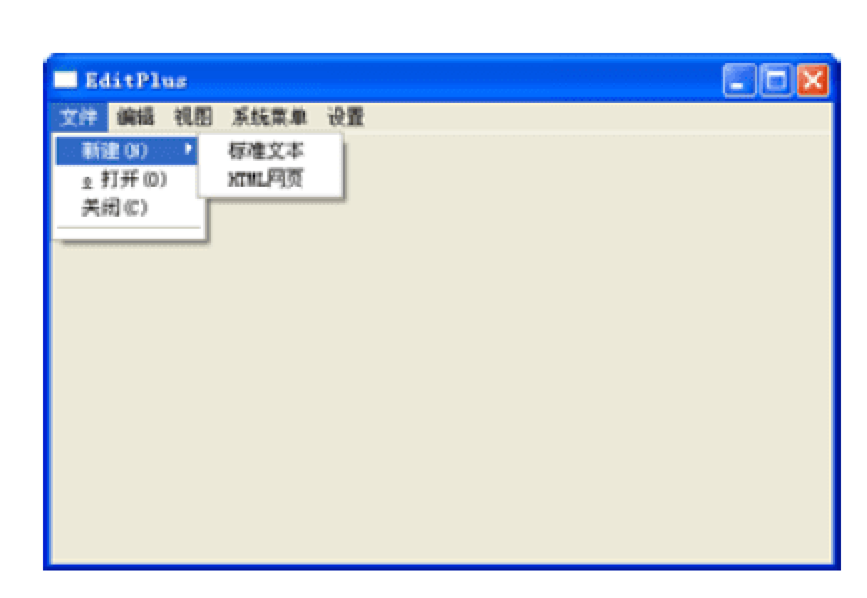


图 14-52　二级菜单

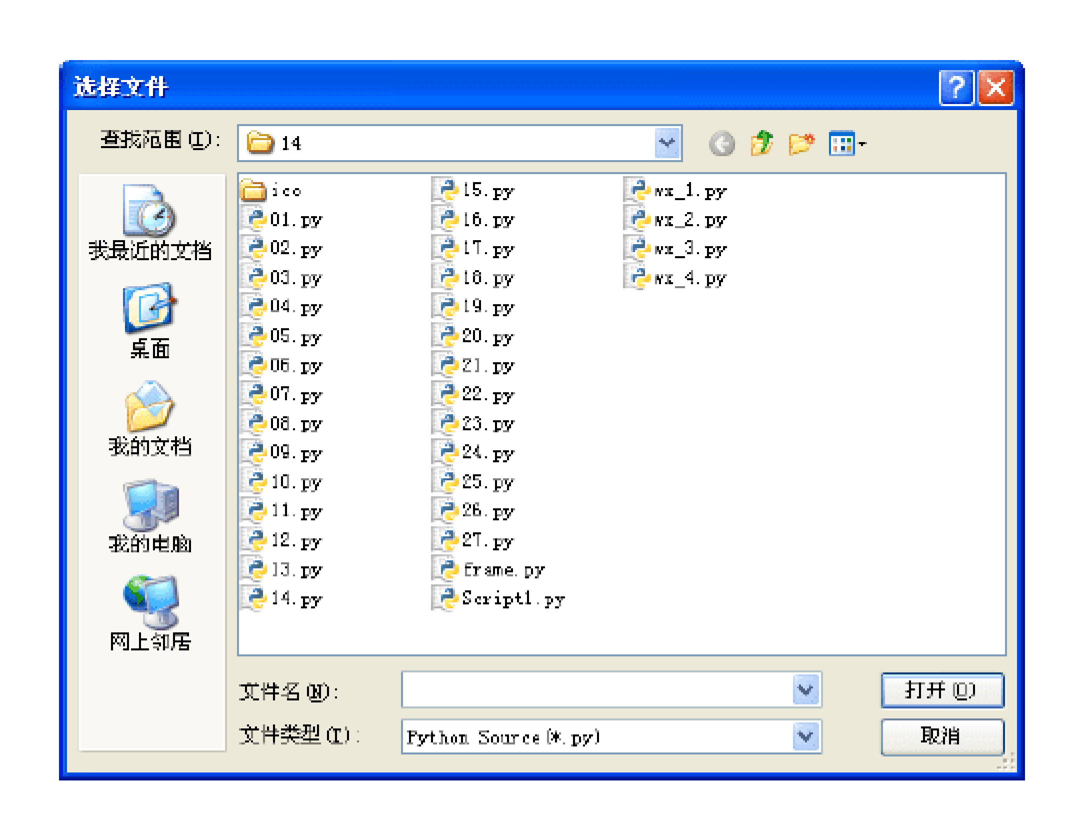


图 14-53　“选择文件”对话框

选择“系统菜单”，弹出“退出”子菜单，将鼠标放在该子菜单项上面，弹出对话框，如图 14-54 所示，这时“退出”菜单项的内容高亮显示。选择“设置”主菜单下的“打开/屏蔽”子菜单项，“系统菜单”下的“退出”子菜单处于禁用状态，如图 14-55 所示。

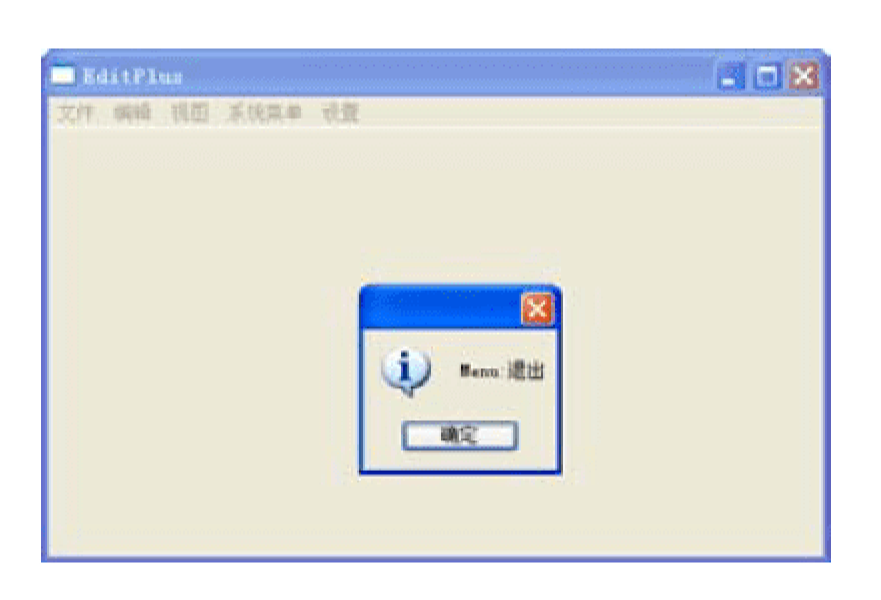


图 14-54　触发EVT_MENU_HIGHLIGHT事件

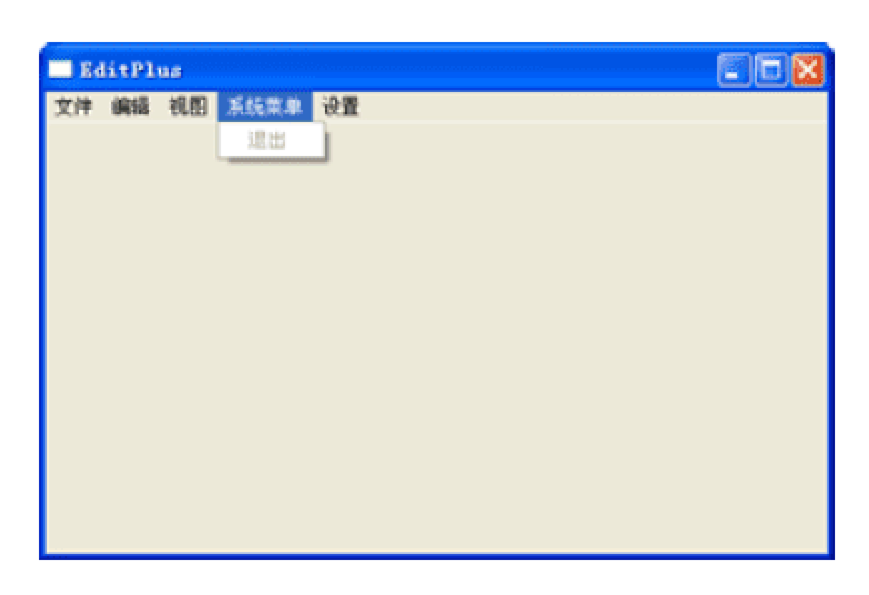


图 14-55　菜单项处于禁用状态

14.5.5　实例分析



源码解析

在该案例中，实现了菜单的 EVT_MENU_HIGHLIGHT 事件，并对子菜单的打开、屏蔽进行控制。当子菜单的 Enable 属性为 True 时，设置为 False；当子菜单 Enable 属性为 False 时，设置为 True。首先通过 MenuBar 对象的 IsEnable()方法获取某个子菜单当前的 Enable 属性，然后传递菜单的 ID 值给 IsEnabled()即可返回相应的布尔值。最后调用 Menu 对象的 Enable()方法对子菜单的 Enable 属性进行设置。

14.6　常见问题解答

14.6.1　应用程序启动时立即崩溃



应用程序启动时立即崩溃，这是怎么回事呀？
网络课堂：http://bbs.itzcn.com/thread-16712-1-1.html

当应用程序启动时，系统立即崩溃，或崩溃后出现一个空白的窗口，这是怎么回事呢？

【解决办法】当启动应用程序时，系统崩溃，原因是在 wx.App 创建之前，已经创建或使用了一个 wxPython 对象。只要在启动脚本时立即创建 wx.App 对象即可。

14.6.2　顶级窗口刚创建便立即关闭



顶级窗口刚刚创建便立即关闭，怎么回事？
网络课堂：http://bbs.itzcn.com/thread-16713-1-1.html

顶级窗口刚刚创建便立刻关闭了，应用程序也立即退出，这是这么回事呀？

【解决办法】造成这种情况的原因可能是没有调用 wx.App 的 MainLoop()方法的问题，只要在所有设置完成后调用 MainLoop()方法即可。

14.7　习　　题

一、填空题

(1) 应用程序对象包含__________方法，它在启动时被调用。在这个方法中，通常要初始化框架和别的全局对象。wxPython 应用程序通常在它的所有顶级窗口被关闭或主事件循环退出时结束。

(2) 在 wxPython 中，可以通过调用 Frame 类的__________方法来生成窗口工具栏。

(3) 在画线处填入__________，运行后的效果如图 14-56 所示。

```
import wx
class MyFrame(wx.Frame):
    def __init__(self):
        wx.Frame.__init__(self,None,-1,'多级菜单',size=(300,100))
        panel=wx.Panel(self)
        menu=wx.Menu()
        sMenu=wx.Menu()
        sMenu.Append(-1,'菜单一')
        sMenu.Append(-1,'菜单二')
        menu.__________(-1,'子菜单',sMenu)
        menuBar=wx.MenuBar()
        menuBar.Append(menu,'菜单')
        self.SetMenuBar(menuBar)
if __name__=='__main__':
    app=wx.PySimpleApp()
    frame=MyFrame()
    frame.Show()
    app.MainLoop()
```



图 14-56　多级菜单

二、选择题

(1) wxPython 程序的实现基于两个必要的对象：__________，任何 wxPython 应用程序都需要实例化一个 wx.App，并且至少有一个顶级窗口。

A. wx.Frame 对象和 wx.App 对象　　B. 应用程序对象和顶级窗口

C. 应用程序对象和 wx.Frame 对象　　D. wx.App 对象和 wx.App 对象

(2) 下面代码画线处应填写__________，可生成带复选框的列表框，如图 14-57 所示。

```
import wx
```

```
class ListBoxFrame(wx.Frame):
    def __init__ (self):
        wx.Frame.__init__(self,None,-1,'列表框',size=(200,200))
        panel=wx.Panel(self,-1)
        langList=['Python','Java','ASP.NET']
        self.checkListBox=wx.___________
(panel,-1,(10,10),(80,100),langList,wx.LB_SORT)
if __name__=='__main__':
    app=wx.PySimpleApp()
    ListBoxFrame().Show()
    app.MainLoop()
```



图 14-57　带复选框的列表框

A. CheckListBox　　B. ListBox　　C. Choice　　D. ComboBox

(3) Sizers 布局管理器的使用一般分为 3 个步骤，下面选项中排列正确的是__________。

A. 创建 Sizers 布局管理器，调用 SetSizer()方法将布局管理器添加到容器中，调用容器的 Add()方法将各个组件添加到布局管理器中

B. 创建 Sizers 布局管理器，创建组件，调用容器的 Add()方法将各个组件添加到布局管理器中，运行程序

C. 创建 Sizers 布局管理器，调用容器的 Add()方法将各个组件添加到布局管理器中，调用 SetSizer()方法将布局管理器添加到容器中

D. 创建 Sizers 布局管理器，调用容器的 Add()方法将各个组件添加到布局管理器中，运行程序

三、上机练习

上机练习：向菜单中添加指定的菜单项。

在窗口的面板中添加一个输入文本框和一个按钮，如图 14-58 所示。当在输入文本框中输入所要添加的菜单项后，单击“添加新菜单项”按钮，则向原有的“文件”菜单中添加输入的菜单项，如图 14-59 所示。

当选择“新建”子菜单时，弹出提示对话框，如图 14-60 所示；当选择新添加的菜单项时，也会弹出相应的提示对话框，如图 14-61 所示；当选择“退出”子菜单时，关闭窗口。



图 14-58　添加菜单窗口

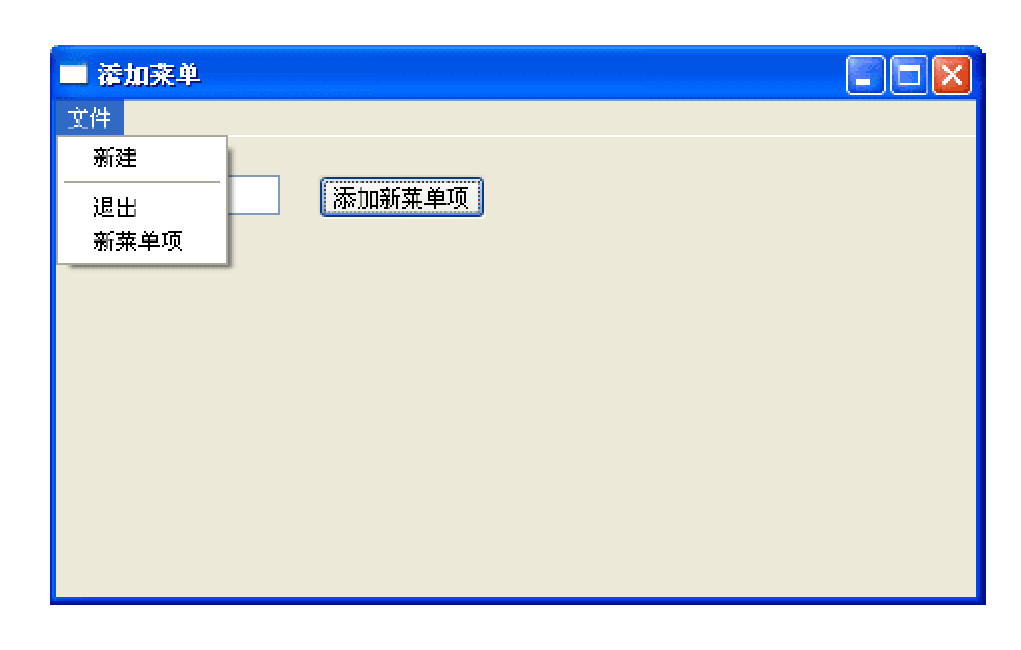


图 14-59　向菜单中添加子菜单



图 14-60　选择“新建”子菜单项

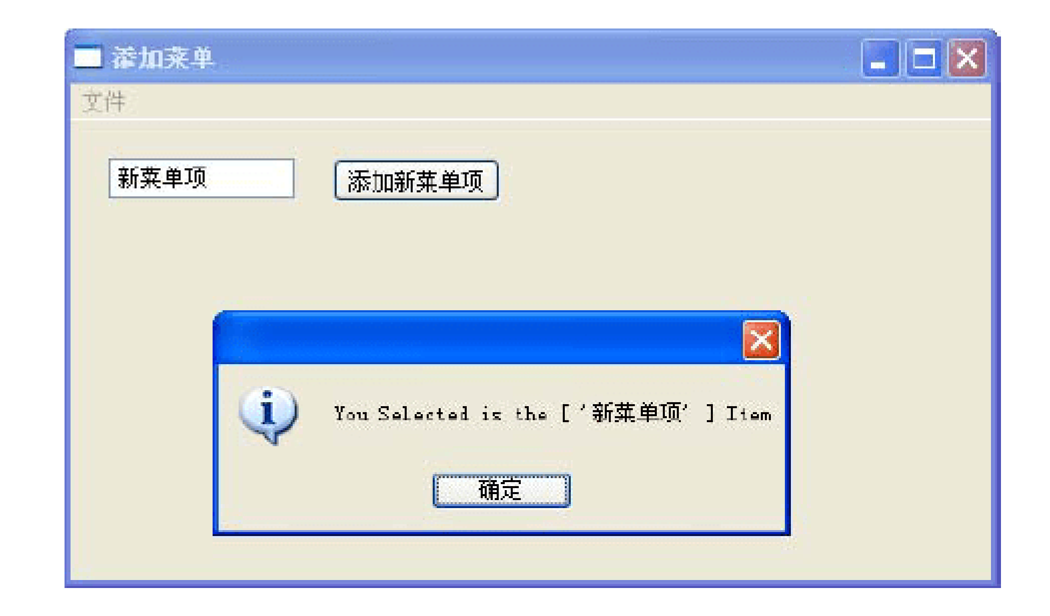


图 14-61　选择新添加的菜单项



第 15 章　Python的Web开发之Django框架应用

内容摘要

自分层结构的 Web 设计理念普及以来，选择适合的开发框架无疑是项目成功的关键。在动态语言领域，Python、Ruby、Groovy 等语言在 Web 开发中的应用日益广泛。Python 语言 Web 框架 Django 以其新颖简洁的开发模式和巨大的发展潜力，逐渐赢得大量开发者的青睐。

本章将详细介绍什么是 Django 框架，Django 框架中的 MVC 模式，以及如何应用 Django 框架及 Django 框架的高级应用。

学习目标

- 了解 Django 框架。
- 了解 Django 框架中的 MVC。
- 掌握 Django 框架的开发环境搭建。
- 掌握 Django 框架的应用。
- 掌握 Django 框架的高级应用。

15.1 Django框架简介

Django 是应用于 Web 开发的高级动态语言框架，最初起源于美国芝加哥的 Python 用户组，具有新闻从业背景的 Adrian Holovaty 是 Django 框架的主要开发者。在 Adrian 的带领下，Django 小组致力于为 Web 开发者贡献一款高效、完美的 Python 开发框架，并且在 BSD(Berkeley Software Distribution，伯克利软件套装)开放源代码协议许可下授权给开发者自由使用。本节将简单介绍 Django 框架的功能以及该框架的优缺点。

**视频教学**：光盘/videos/15/ Django 框架简介.avi　　长度：4 分钟

Django 拥有完善的模板机制、对象关系映射机制以及用于动态创建后台管理界面的功能。使用 Django 框架来开发 Web 应用，可以快速设计和开发具有 MVC 层次的 Web 应用。Django 框架是从实际项目中诞生出来的，该框架提供的功能特别适合于动态网站的建设，特别是管理接口。

在 Django 框架中，包含了开发 Web 网络应用所需的组件。这些组件包括数据库的对象关系映射、动态内容管理的模板系统和丰富的管理界面。在 Django 框架中，可以使用管理脚本文件 manage.py 来构建简单的开发服务器。当然，同样可以使用 mob_python 或者 FastCGI 模块来构建。

Django 框架作为一种快速的网络框架，具有下面的一些特点。

- 组件的合理集成：在 Django 框架中，已经有一套集成在一起的组件。这些组件都是 Django 项目组开发的，并为开源界所修改和使用。Django 框架中组件的设计目的是实现重用性，并具有易用性。
- 对象关系映射和多数数据库支持：Django 框架的数据库组件——对象关系映射(Object-Relation Mapper，ORM)提供了数据模块和数据引擎之间的接口。支持的数据库包括 PostgreSQL、MySQL 和 SQLite 等。这种设计使得在切换数据库的时候只需要修改配置文件即可。这为应用开发者在设计数据库时提供了很好的灵活性。
- 简洁的 URL 设计：Django 框架中的 URL 系统设计非常强大而灵活。可以在 Web 应用中为 URL 设计匹配模式，并用 Python 函数处理。这种设计使得 Web 应用的开发者可以创建与用于友好的 URL，同时对搜索引擎也是友好的。
- 自动化的管理界面：在 Django 框架中已经提供了一个易用的管理界面，这个界面可以方便地管理用户数据，具有高度的灵活性和可配置性。
- 强大的开发环境：在 Django 中提供了强大的 Web 开发环境，其中有一个可用于开发和测试的轻量级 Web 服务器。当启用调试模式后，Django 会显示大量的调试信息，使得消除 BUG 非常容易。

15.2　MVC模式

MVC(Model-View-Controller 的缩写)是一种在软件工程中广泛使用的设计模式。MVC 应用程序由三部分组成：模型(Model)、视图(View)和控制器(Controller)，即把一个应用的输入、处理和输出流程按照 Model、View 及 Controller 的方式进行分离，使得一个应用被分成 3 个层——模型层、视图层和控制层。

 视频教学：光盘/videos/15/ MVC 模式.avi　　　　长度：14 分钟

15.2.1　基础知识——MVC模式介绍

MVC 模式将一个应用分为 3 个层：模型层、视图层和控制层。通过这种分层方式，设计模式可以将应用的输入处理、界面显示以及控制流程分开，开发者可自由地修改应用的输入处理而不用担心界面显示问题。也就是说，MVC 模式可以有效地将应用分为不同的模块，从而实现应用的松耦合。

1. 模型层

模型层负责业务流程和状态的处理以及业务规则的制定。业务流程的处理过程对其他层来说是暗箱操作，模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 的核心。目前流行的 EJB(Enterprise Java Beans，Java 企业 Bean)模型就是一个典型的应用，它从应用技术角度对模型做了进一步划分，以便充分利用现有的组件，但它不能作为应用设计模型的框架。它仅仅告诉你按这种模型设计就可以利用某些技术组件，从而减少了技术上的困难。对开发者来说，可以专注于业务模型的设计。MVC 设计模式告诉我们，把应用的模型按一定的规则抽取出来，抽取的层次很重要，这也是判断开发人员是否优秀的依据。抽象与具体不能隔得太远，也不能太近。MVC 没有提供模型的设计方法，只告诉你应该组织管理这些模型，以便模型的重构和提高重用性。我们可以用对象编程来做比喻，MVC 定义了一个顶级类，告诉它的子类只能做这些，但没法限制你能做这些。这对编程人员非常重要。

业务模型还有一个很重要的模型就是数据模型。数据模型主要指实体对象的数据保存(持续化)。比如将一张订单保存到数据库，然后从数据库获取订单。我们可以将这个模型单独列出，而且所有有关数据库的操作只限制在该模型中。

2. 视图层

视图代表用户交互界面，对 Web 应用来说，可以概括为 HTML 界面，当然也可能为 XHTML、XML 和 Applet。随着应用的复杂性和规模性，界面处理也变得具有挑战性。一个应用可能有很多不同的视图，MVC 设计模式对视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括视图上的业务流程处理。业务流程处理交予模型(Model)处理。比如一个订单的视图只接受来自模型的数据并显示给用户，以及将用户界面的输入数据和请求传递给控制和模型。

3. 控制层

控制层可以理解为从用户接收请求，将模型与视图匹配在一起，共同完成用户的请求。控制层的作用就是一个分发器，负责将具体的数据转发给特定的业务逻辑处理，并将处理后的数据输出。控制层并不做任何数据处理的工作。例如，用户单击一个超链接，控制层接受请求后并不处理业务信息，它只把用户的信息传递给模型，告诉模型做什么，然后选择符合要求的视图返回给用户。因此，一个模型可能对应多个视图，一个视图也可能对应多个模型。

模型、视图与控制器的分离，使得一个模型可以具有多个视图。如果用户通过某个视图的控制器改变了模型的数据，那么所有其他依赖于这些数据的视图都应反映这些变化。无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，使显示及时更新。这实际上是一种模型的变化与传播机制。模型、视图和控制器三者之间的关系和各自的主要功能，如图 15-1 所示。

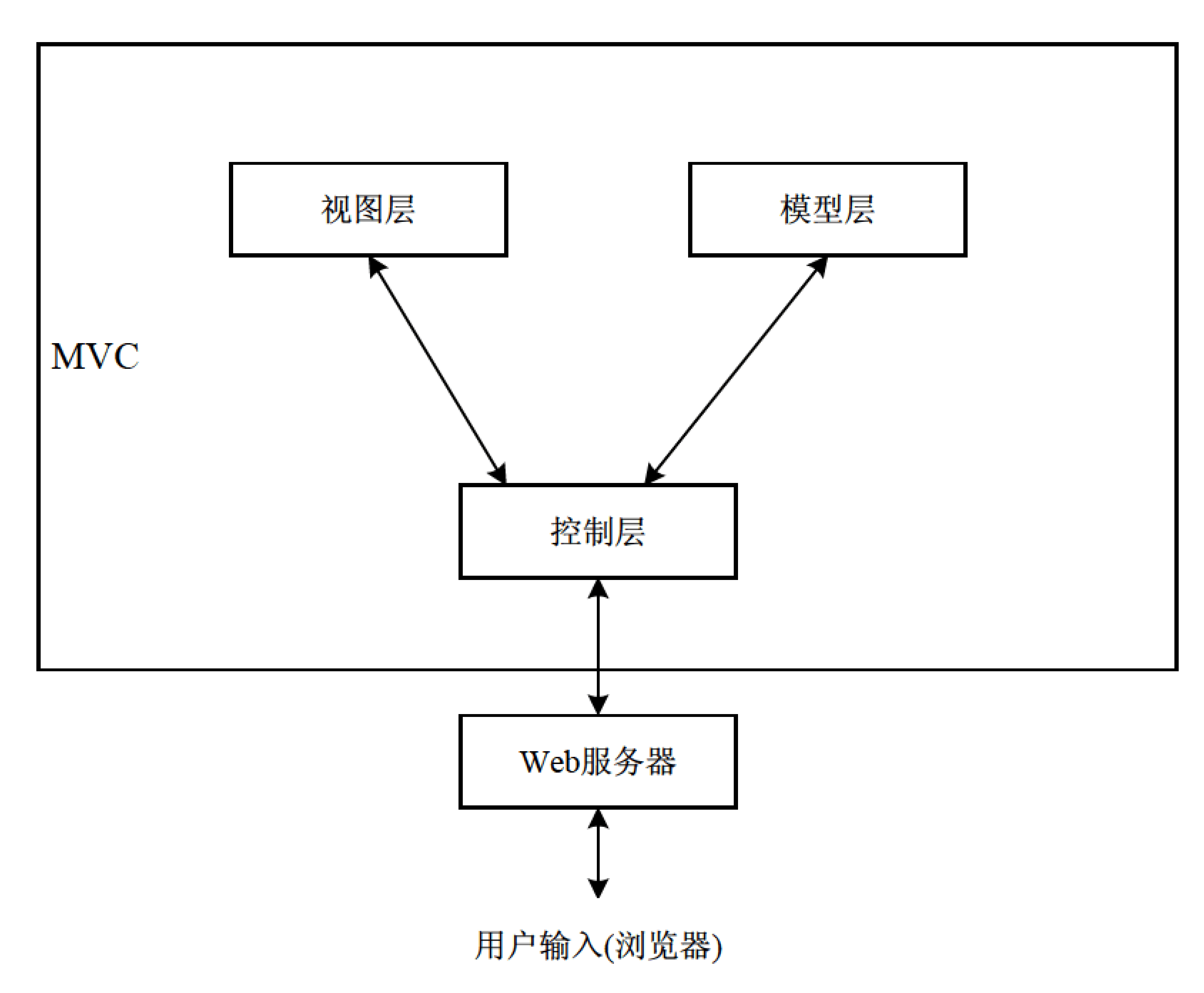


图 15-1 Web应用的MVC模式

15.2.2 基础知识——MVC模式的优点和缺点

金无足赤，人无完人。MVC 模式有哪些优点和缺点呢？下面具体分析一下。

1. MVC模式的优点

- 可以为一个模型在运行的同时建立和使用多个视图。变化与传播机制可以确保所有相关视图及时得到模型数据变化，从而使所有关联的视图和控制器做到行为同步。
- 视图与控制器的可接插性，允许更换视图和控制器对象，而且可以根据需求动态地打开或关闭，甚至在运行期间进行对象替换。

- 模型的可移植性。因为模型是独立于视图的，所以可以把一个模型独立地移植到新的平台上工作，需要做的只是在新平台上对视图和控制器进行修改。
- 潜在的框架结构。可以基于此模型建立应用程序框架，不仅仅用在界面的设计中。

2. MVC模式的缺点

- 增加了系统结构和实现的复杂性。对于简单的界面，严格遵循 MVC，使模型、视图与控制器分离，会增加结构的复杂性，可能产生过多的更新操作，从而降低运行效率。
- 视图与控制器间过于紧密的连接。视图与控制器相互分离，但联系非常紧密的部件，视图没有控制器的存在，其应用很有限，反之亦然，这样就妨碍了它们的独立重用。
- 视图对模型数据的低效率访问。依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。
- 目前，一般高级的界面工具或构造器不支持 MVC 模式。改造这些工具以适应 MVC 的需要和建立分离的部件的代价非常大，从而造成使用 MVC 的困难。

MVC 的分层方法可以将业务逻辑和视图显示分开，从而实现简化处理加速开发的目的。这样的设计使得最终的应用结构清晰，扩展性强。



使用 MVC 模式可以有效地构建 Web 应用框架。由于这种模块化的构成，使得多个视图对应到一个模型。这种做法的一个最大的好处是，可以迅速实现用户的需求。例如在数据模型中可能有很多相似类型的数据，可能需要输出为 HTML 显示，或者 XML 显示，但是对数据的处理则可能是类似的。如果遵循 MVC 模式设计，则可以使用一个数据模型来实现。通过这种设计，可以减轻代码的维护负担。当页面输出变化的时候，不需要修改数据模型。

15.2.3　基础知识——Django框架中的MVC

Django 作为一个流行的基于 Python 的 Web 开发框架，也支持 MVC 模式。在 Django 框架中，当 URL 被请求时，将会调用指定的 Python 方法。通过业务逻辑处理后，将会通过模板来呈现页面。在 Django 的设计小组中，将这种实现方式成为 MVT(Model-View-Template)框架。

在数据模型中，Django 使用 django.db.model.Model 实现了网站设计中需要使用的数据模型。在数据模型中定义了保存在数据库中的各种对象及其属性。通过继承自 Model 类生成的对象，可以通过添加 Field 来为特定的数据增加方法。在 Django 数据模型中，提供了丰富的访问数据对象接口。数据模型中的数据将会同步到后台的数据库，而 Django 则提供了一个良好的 ORM(Object-Relation Mapping，对象-关系映射)，使得开发者可以从视图和模板中访问数据库中的数据。

在视图层中，Django 框架实现了良好的 URL 设计和处理。当收到 URL 请求时，Django 将会使用一组预订的 URL 模式来匹配到合适的处理器。实际上，URL 的设计也是网站视图层设计，决定了 Web 应用如何读取 URL 请求以及如何显示网页。对每个特定的 URL，Django 都会有一个特定的视图函数来进行处理。可以看到，Django 框架的视图处理分成多个步骤进行。其框架在收到 URL 请求后，通过页面函数来处理，最后将页面响应返回给浏览器(客户端)显示。

在 Django 框架中，还提供了强大的模板解析功能，通过页面函数来输出页面响应。模板

系统使得 Web 应用的开发者集注意力于需要展现的数据上，这种开发方式使得页面设计者只需关注与输出网页的构成。

Django 框架良好的 MVC 模式设计使得 Django 成为一个流行的 Web 开发框架。

15.3　Diango开发环境的搭建

在使用 Django 框架之前，首先要搭建 Django 开发环境。而要搭建开发环境，首先需要下载 Django 软件包并安装。安装完毕后，还需要设置数据库环境。本节将详细介绍 Django 框架的环境搭建过程。


视频教学：光盘/videos/15/Diango 开发环境的搭建.avi　　**长度：**6 分钟

15.3.1　基础知识——Django框架的安装

由于 Python 语言可以跨平台，因此 Django 可以很方便地被安装在包括 Windows 和 Linux 等系统平台上。因为 Django 框架是使用 Python 语言开发的，且框架中已经实现了 Web 开发所需的组件，所以安装 Django 框架的基本条件是系统中已经安装了 Python。

(1) 打开 Django 框架的下载页面 http://www.djangoproject.com/download，将 Django 源码包 Django-1.3.tar.gz 下载到本地。

(2) 将下载的 Django-1.3.tar.gz 解压为 Django-1.3。

(3) 在 Windows 命令行中，转到 Django-1.3 目录下，如图 15-2 所示。

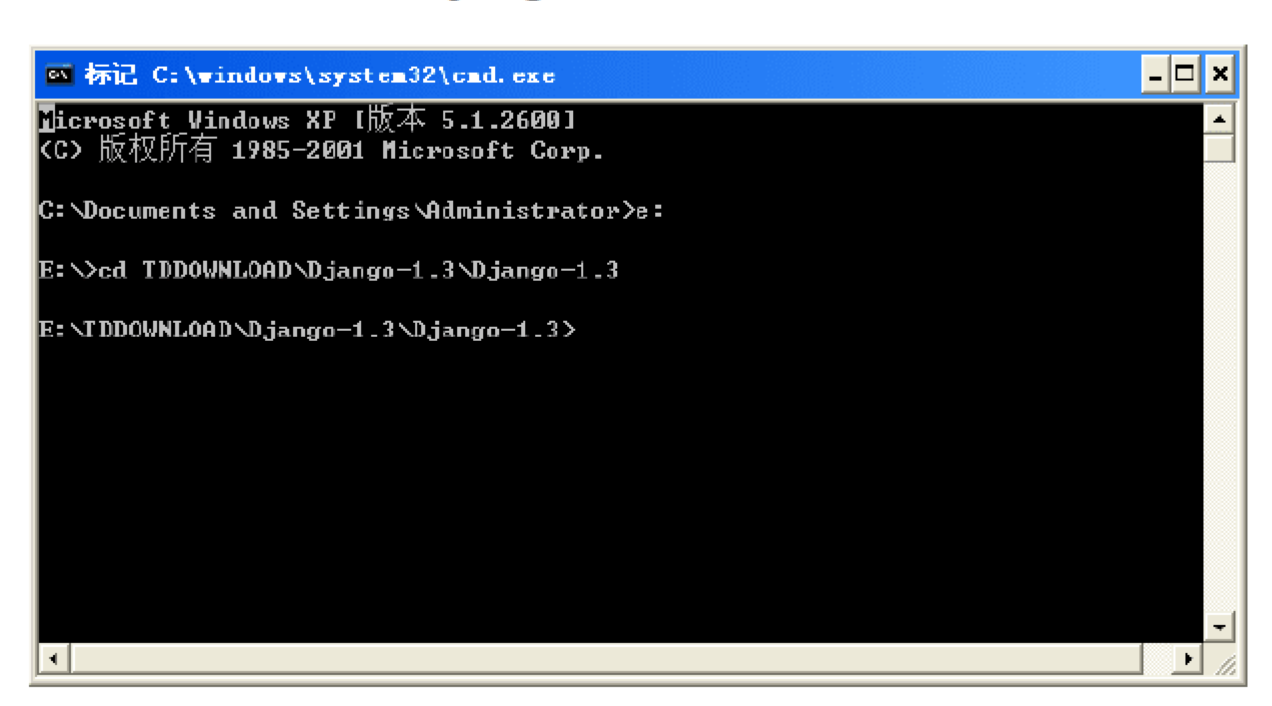


图 15-2　转到Django源码包目录

(4) 运行命令 E:\TDDOWNLOAD\Django-1.3\Django-1.3>setup.py install，进入 Django 框架的安装程序，自动安装。

对于特定的系统平台，可以安装针对特定平台的软件包。例如在 Ubuntu 和 Debian 等发行版的 Linux 中，可以使用 apt 程序来安装。

```
apt-get install python-django
```

安装完成后，打开 Python UGI，输入以下代码：

```
>>> import django
>>> print django.VERSION
```

按回车键，输出：

```
(1, 3, 0, 'final', 0)
```

如果返回信息为安装的 Django 版本，则表示已经安装成功。

15.3.2　基础知识——数据库的配置

Django 框架的唯一需求是结合 Python，所以数据库在 Django 的 Web 开发中并不是至关重要的，但是在实际的 Web 设计中，大部分数据仍然保存在数据库中，而 Django 支持多种数据库，包括 MySQL、PostgreSQL 和 SQLite 等。Django 有着一个设计良好的 ORM，可以有效地屏蔽底层数据库的不同。

对于每个 Django 应用，其目录中都有一个 setting.py 文件，用来实现对数据库的配置。setting.py 文件是一个 Python 脚本文件，在其中可以设置 Django 项目的属性。每个 Django 项目都有特定的配置文件。

在 setting.py 文件中，可以通过设置下面的属性值来控制 Django 对数据库的访问。

- DATABASE_ENGINE：设置数据库引擎的类型，可以设置的类型有 sqlite3、mysql、postgresql 和 ado_mssql 等。
- DATABASE_NAME：设置数据库的名称。如果数据库引擎使用的是 SQLite，则需要指定全路径。
- DATABASE_USER：指定连接数据库时的用户名。当数据库引擎使用 SQLite 时，不需要设置此值。
- DATABASE_PASSWORD：指定用户 DATABASE_USER 的密码。当数据库引擎使用 SQLite 时，不需要设置此值。
- DATABASE_HOST：指定数据库所在的主机。当此值为空时，表示数据将保存在本机中。当数据库引擎引用 SQLite 时，不需要设置此值。
- DATABASE_POST：设置连接数据库时使用的端口号。当不设置该值时，将使用默认端口号。当数据库引擎使用 SQLite 时，不需要设置此值。

15.4　使用Django框架制作通讯录

当 Diango 开发环境搭建成功之后，下面就可以使用 Diango 框架开发 Web 应用了。Diango 框架提供了一种迅速的方法来创建功能丰富的 Web 应用。本节将描述一个 Web 应用创建的完整过程。

视频教学：光盘/videos/15/ Web 应用的创建.avi　　长度：19 分钟

视频教学：光盘/videos/15/创建数据模型.avi　　长度：11 分钟

15.4.1 基础知识——Web应用的创建

在 Django 框架中，一个网站可以使用多个 Django 项目来构成，而一个 Django 项目则包含一组特定的对象，这些对象包括 URL 设计、数据库设计以及其他一些选项设置。Django 框架使用 django-admin.py 文件对 Web 应用进行管理。当 Django 软件包安装完毕，在 Scripts 目录中将会包含 django-admin.py 文件。另外，如果是在 Linux 下使用安装包安装，将会创建 django-admin 链接。

django-admin.py 文件中有许多命令选项，可以通过这些选项来对项目进行操作。运行 django-admin.py 文件，将输出 Django 支持的所有命令选项。

```
Usage: django-admin.py subcommand [options] [args]
Options:
  -v VERBOSITY, --verbosity=VERBOSITY
                        Verbosity level; 0=minimal output, 1=normal output,
                        2=all output
  --settings=SETTINGS   The Python path to a settings module, e.g.
                        "myproject.settings.main". If this isn't provided, the
                        DJANGO_SETTINGS_MODULE environment variable will be
                        used.
  --pythonpath=PYTHONPATH
                        A directory to add to the Python path, e.g.
                        "/home/djangoprojects/myproject".
  --traceback           Print traceback on exception
  --version             show program's version number and exit
  -h, --help            show this help message and exit

Type 'django-admin.py help <subcommand>' for help on a specific subcommand.

Available subcommands:
  cleanup
  compilemessages
  createcachetable
  dbshell
  diffsettings
  dumpdata
  flush
  inspectdb
  loaddata
  makemessages
  reset
  runfcgi
  runserver
  shell
  sql
  sqlall
  sqlclear
  sqlcustom
  sqlflush
  sqlindexes
  sqlinitialdata
  sqlreset
  sqlsequencereset
  startapp
  startproject
```

```
  syncdb
  test
  testserver
  validate
```

从输出结果可以看出，Django 框架中有大量的命令选项用来管理 Django 项目。如果需要查看特定命令选项的功能，可以使用 django-admin.py help startproject 来完成。在 Windows 命令行中转到 django-admin.py 文件所在的目录，然后输入：

```
django-admin.py help startproject
```

按回车键，将输出 Django 框架中的特定命令选项。

```
Usage: django-admin.py startproject [options] [projectname]
Creates a Django project directory structure for the given project name in the c
urrent directory.
Options:
  -v VERBOSITY, --verbosity=VERBOSITY
                    Verbosity level; 0=minimal output, 1=normal output,
                    2=all output
  --settings=SETTINGS   The Python path to a settings module, e.g.
                    "myproject.settings.main". If this isn't provided, the
                    DJANGO_SETTINGS_MODULE environment variable will be
                    used.
  --pythonpath=PYTHONPATH
                    A directory to add to the Python path, e.g.
                    "/home/djangoprojects/myproject".
  --traceback         Print traceback on exception
  --version           show program's version number and exit
  -h, --help          show this help message and exit
```

还可以通过 startproject 来迅速创建项目。在下面的操作中，将会创建一个名称为 Django_Pro 的项目。

```
django-admin.py startproject Django_Pro
```

执行该语句，在 django-admin.py 所在的目录中将会创建一个名称为 Django_Pro 的目录，进入该目录，会看到 4 个文件，它们是一个基本的 Web 应用所必需的，如图 15-3 所示。

下面具体介绍各文件的功能。

- __init__.py：空文件，主要用来指定 Python 语言将此网站目录当作 Python 的包。
- manage.py：可以使网站管理员来管理 Django 项目。
- setting.py：此 Django 项目的配置文件。
- urls.py：包含 URL 的配置文件，这也是用户访问 Django 应用的方式。

这 4 个文件中仅仅包含一个最简单的 Web 应用所需的代码。当应用变得复杂时，将会对这些代码进行扩充。

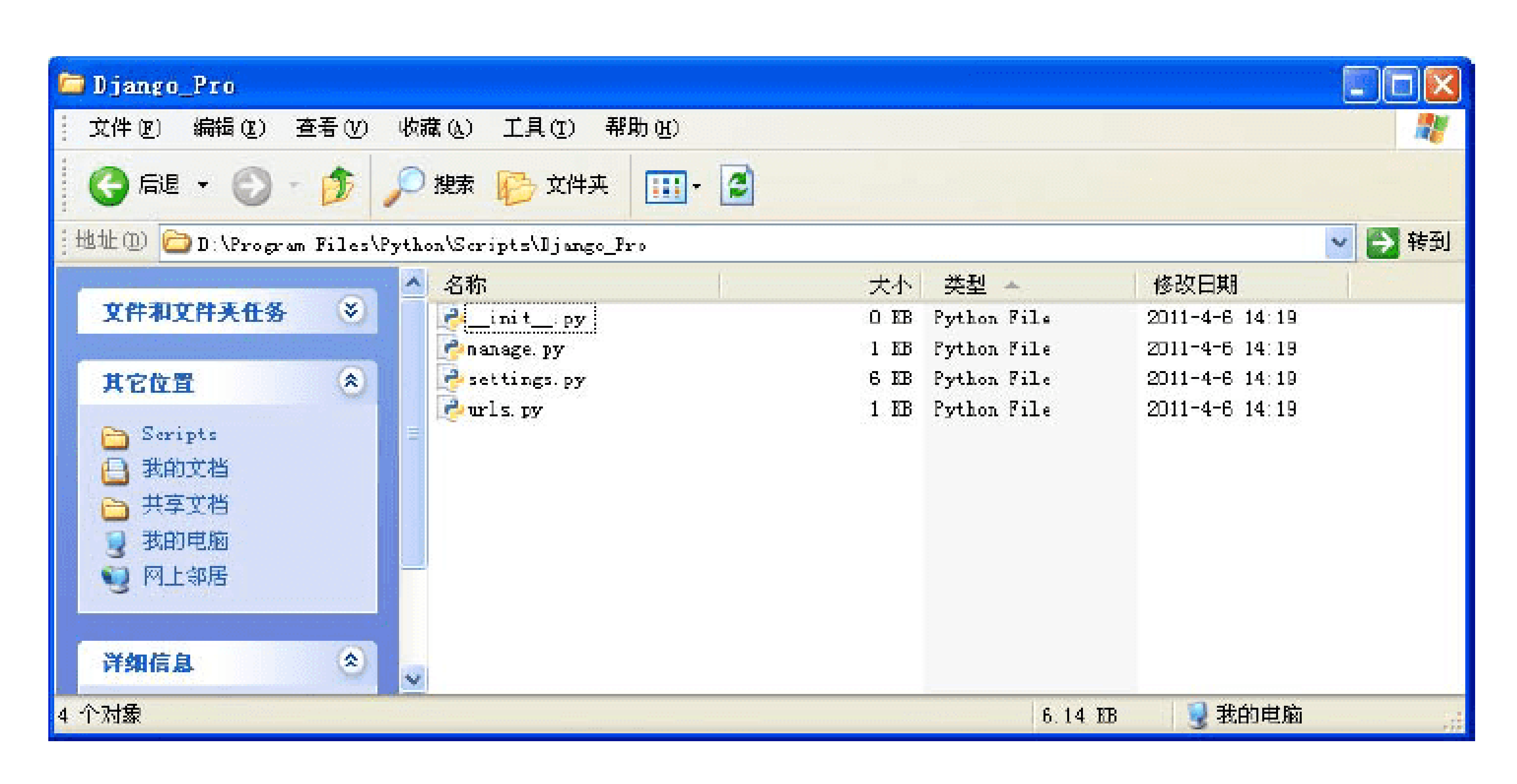


图 15-3 Django框架生成的项目



由于 Django 项目是作为 Python 的包来处理的，因此在项目命名的时候尽量不要和已有的 Python 模块中的名称相冲突，否则在实际使用时可能会出错。另外，尽量不要将网站的代码放在 Web 服务器的根目录下，可能带来安全方面的问题。

15.4.2 基础知识——Django的开发服务器

在 Django 框架中包含了一个轻量级的 Web 应用服务器，可以在开发时使用。在开发 Web 项目时，不需要再对其配置服务器，比如针对 Apache 的配置。Django 提供了内置的服务器，可以在代码修改时自动加载，从而实现网站的迅速开发。

在 Windows 命令行中切换到 Django_Pro 项目的目录，然后使用下面的方法来启动内置的服务器。

```
manage.py runserver
```

按回车键，输出如下：

```
Validating models...
0 errors found
Django version 1.3, using settings 'Django_Pro.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

默认情况下，当使用 manage.py runserver 命令来启动内置服务器时，将会默认使用本机的 8000 端口监听。当 8000 端口被占用时则需要使用其他(比如 8001)端口来监听，此时可以使用下面的命令来监听其他端口。

```
manage.py runserver 8001
```

在上面的命令中，只是在本机进行监听。也就是说，Django 只是接受来自本机的连接。在多人开发 Django 项目的情况下，可能需要从其他主机来访问 Web 服务器，此时可以使用下面的命令来接受来自其他主机的请求。

```
manage.py runserver 0.0.0.0:8000
```

该语句表示对本机的所有网络接口监听 8000 端口，这样可以满足多人合作开发和测试

Django 项目的需求，同时也可以使用其他主机来访问此 Web 服务器。



当启动内置的 Web 应用服务器时，Django 会检查配置的正确性。如果配置正确，则将使用 setting.py 文件中的配置启动此开发服务器。

启动浏览器，连接此 Web 服务器，可以显示 Django 项目的初始化页面，如图 15-4 所示。

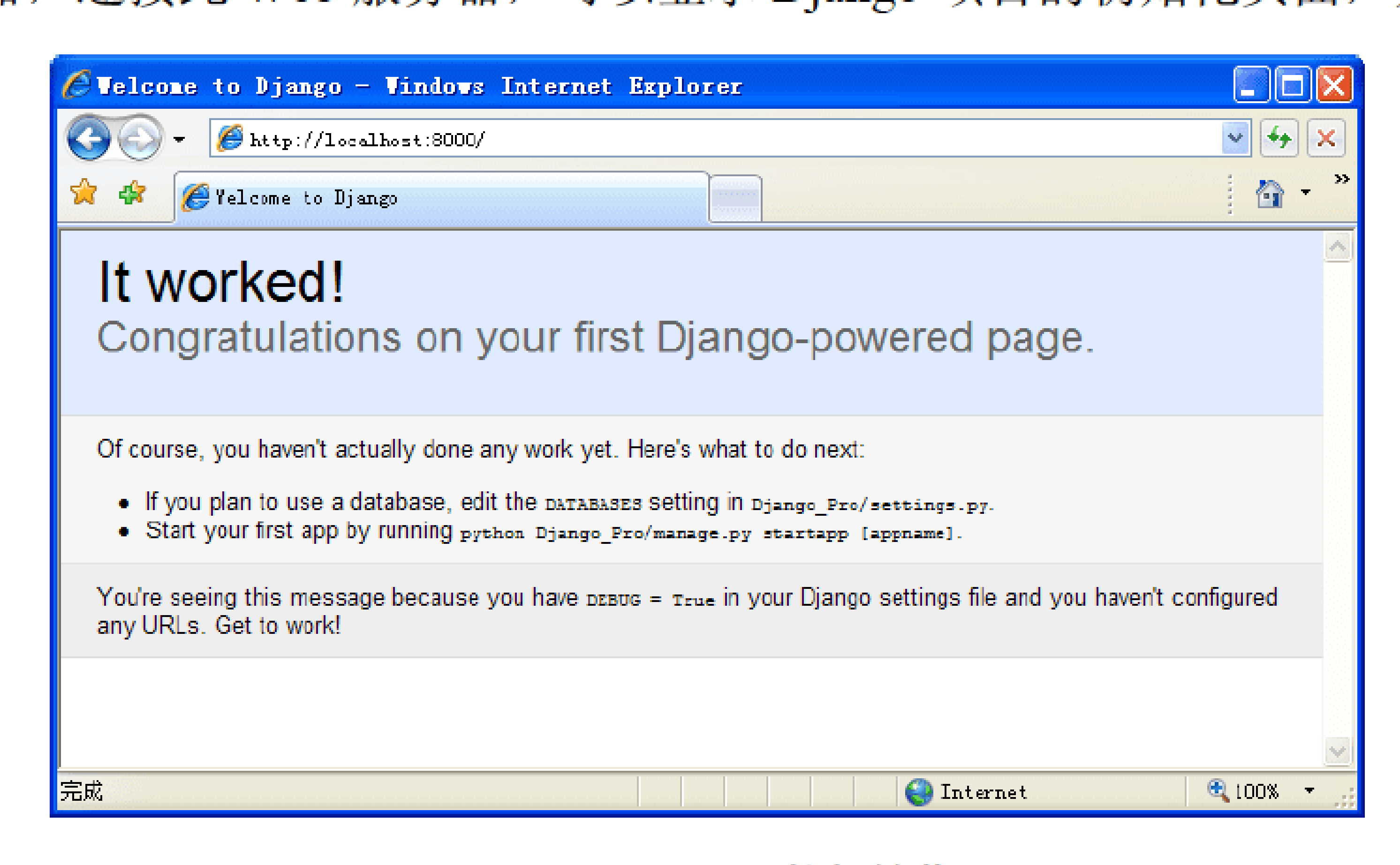


图 15-4　Django项目的初始化页面

当在浏览器的地址栏中输入 http://localhost:8000/时，出现图 15-4 所示的页面，则说明 Django 框架已经正确安装并已经生成一个项目。连接此服务器时，在控制台打印了如下的信息：

```
[06/Apr/2011 14:46:02] "GET / HTTP/1.1" 200 2059
```

这个输出信息显示了连接的时间以及响应信息。在输出响应中，显示了 HTTP 的状态码为 200，表示此连接已经成功。

如果需要中断此服务器，可以按快捷键 Ctrl+C 或者 Ctrl+Break 来完成此操作。

15.4.3　基础知识——创建数据库

前面介绍了如何在 Django 项目中配置和使用数据库。在 Web 开发中，开发者可以选取与自己的项目相吻合的数据库(比如大型的网站需要使用 Orcale 数据库)。SQLite 数据库作为一种轻量级嵌入型的数据库引擎，有着其他数据库所不具备的优点。虽然它只是一个轻量级的数据库引擎，但是已经支持大多数的 SQL 语法。SQLite 由于去除了数据库设计中的复杂部分，使得其具有高效性。SQLite 利用高校的内存组织，将数据库中的数据保存在文件中，使得最后的数据尺寸比其他数据库系统都要小。正是因为 SQLite 使用方便和无需配置的特性，所以本章中的案例将使用 SQLite 数据库引擎。

创建 SQLite 数据库时，首先需要修改 setting.py 文件中 DATABASES 字典，配置相应的属性值来对数据库进行设置。修改 DATABASES 字典如下：

```
DATABASES = {
    'default': {
        'ENGINE': 'sqlite3', # Add 'postgresql_psycopg2', 'postgresql', 'mysql',
'sqlite3' or 'oracle'.
```

```
        'NAME': './djangopro.db3',   # Or path to database file if using sqlite3
        'USER': '',                  # Not used with sqlite3
        'PASSWORD': '',              # Not used with sqlite3
        'HOST': '',      # Set to empty string for localhost. Not used with sqlite3
        'PORT': '',      # Set to empty string for default. Not used with sqlite3
    }
}
```

配置 SQLite 数据库只需要设置两个值：一个是 ENGINE，用来指定使用的是 sqlite3 类型的数据库；另一个是 NAME，用来指定要使用的数据库文件为 djangopro.db3。

接着生成数据库。生成数据库需要通过 manage.py 中的子命令 syncdb 来实现。

```
D:\Program Files\Python\Scripts\Django_Pro>manage.py syncdb
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth user user permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
You just installed Django's auth system, which means you don't have any superuse
rs defined.
Would you like to create one now? (yes/no): yes
Username (Leave blank to use 'administrator'): admin
E-mail address: admin@qq.com
Password:
Password (again):
Superuser created successfully.
Installing custom SQL ...
Installing indexes ...
```

当执行完 syncdb 命令后，在 Django_Pro 目录下会生成一个名称为 djangopro.db3 的文件。在输出结果的前半部分，在数据库中创建了特定的应用，这从配置文件(setting.py)中可以看出。

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
)
```

在输出结果的后半部分，Django 为项目创建了一个管理用户，用户名和密码均为 admin，但在 Windows 命令下的 password 不显示。在接下来的命令行中提示此管理用户的相关信息。

打开 SQLiteManager 管理工具，选择 Open a Database 单选按钮，如图 15-5 所示。

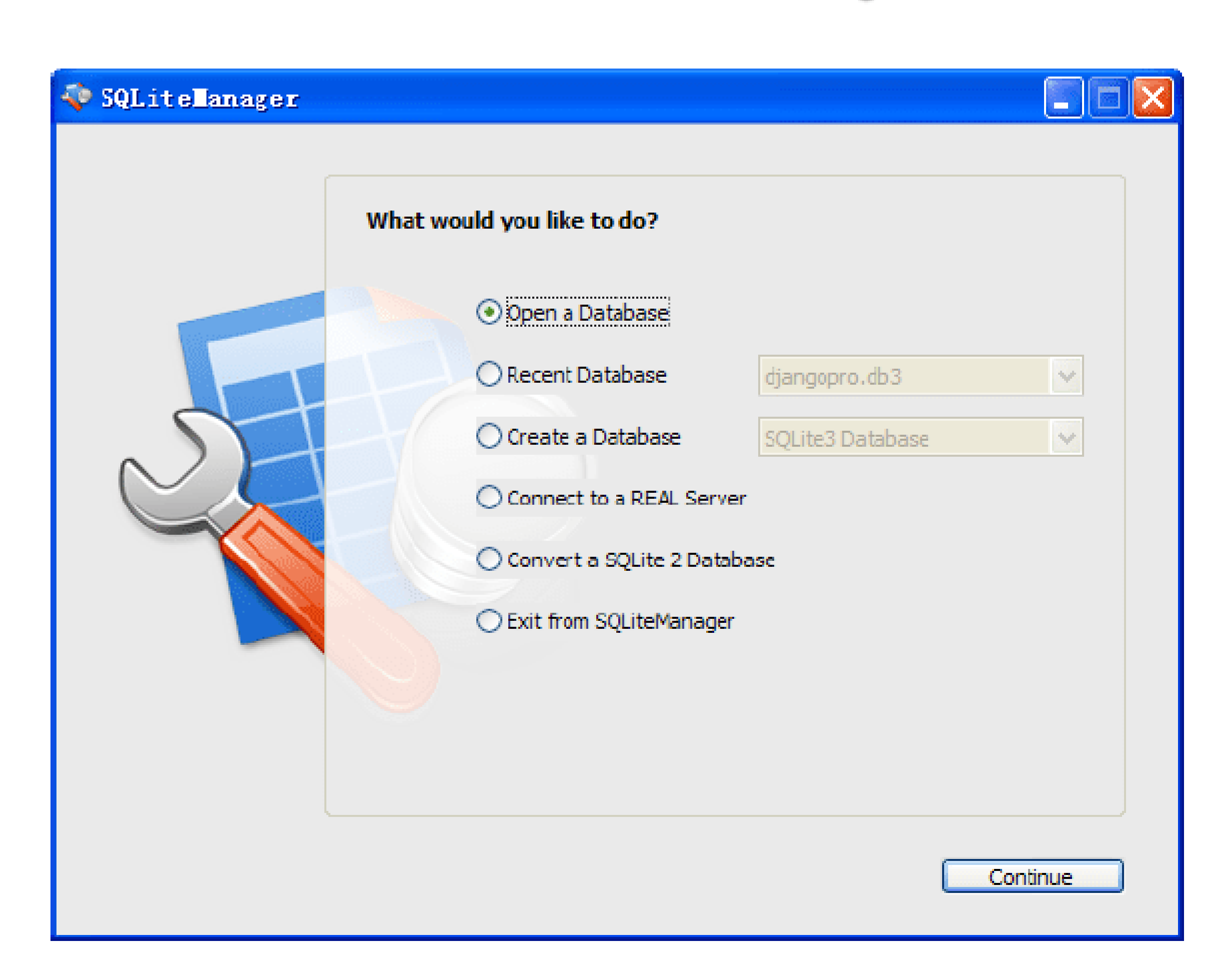


图 15-5　SQLiteManager管理工具选项界面

单击 Continue 按钮，打开选择文件对话框，选择 Django_Pro 目录下的 djangopro.db3 文件，将会看到图 15-6 所示的结果。

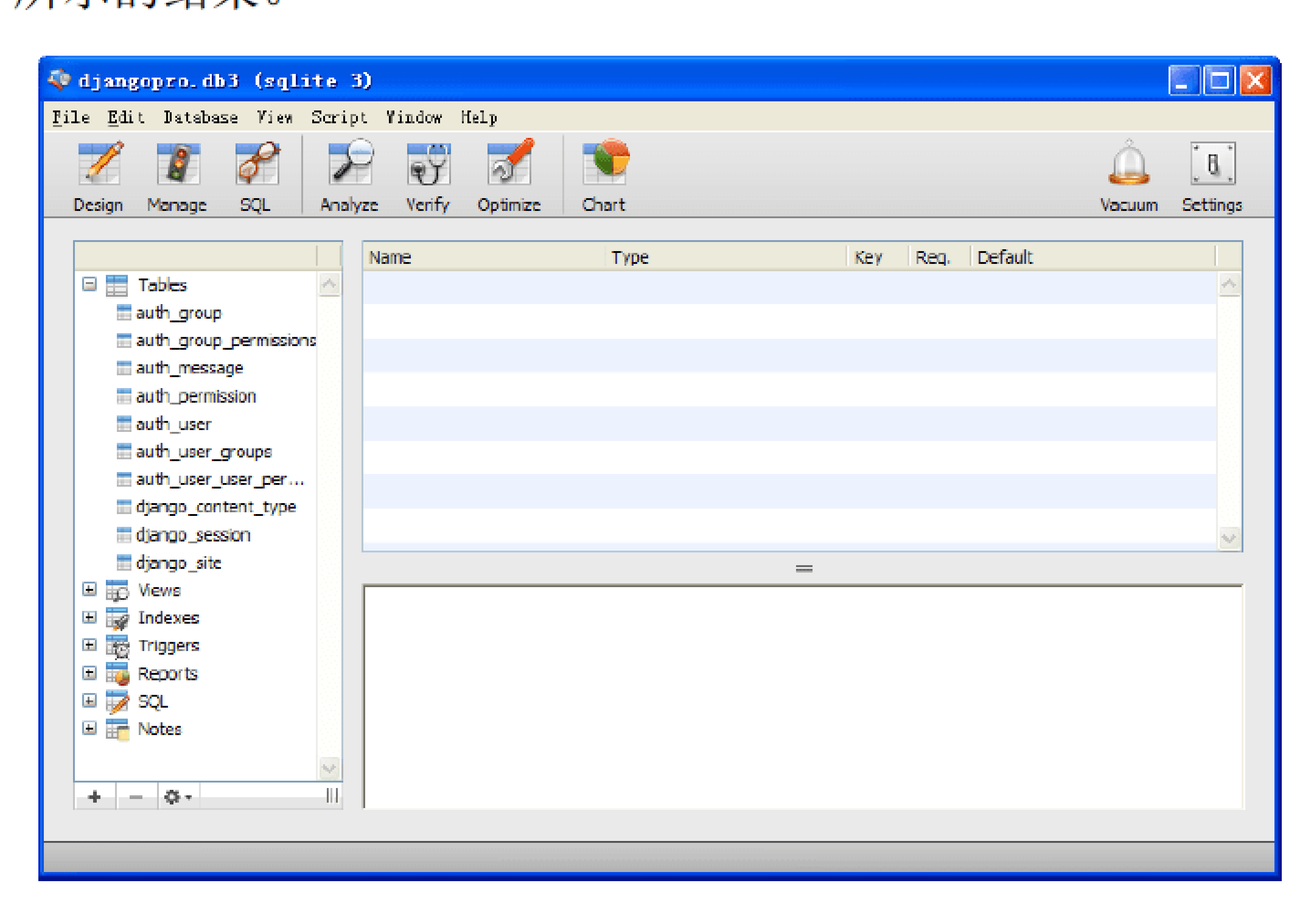


图 15-6　在Django中创建的数据库

15.4.4　基础知识——生成Django应用

一个使用 Django 框架创建的网站，可能会有多个 Django 应用，可以使用 manage.py 文件的 startapp 子命令来生成 Django 应用。一个应用中可以包含一个数据模型以及相关的处理逻辑。在 Windows 命令中切换到 Django_Pro 目录，然后输入：

```
manage.py startapp Users
```

使用 startapp 子命令后，将会在 Django_Pro 目录下生成一个 Users 目录，此目录中的文件定义了应用的数据模型以及处理方式，如图 15-7 所示。

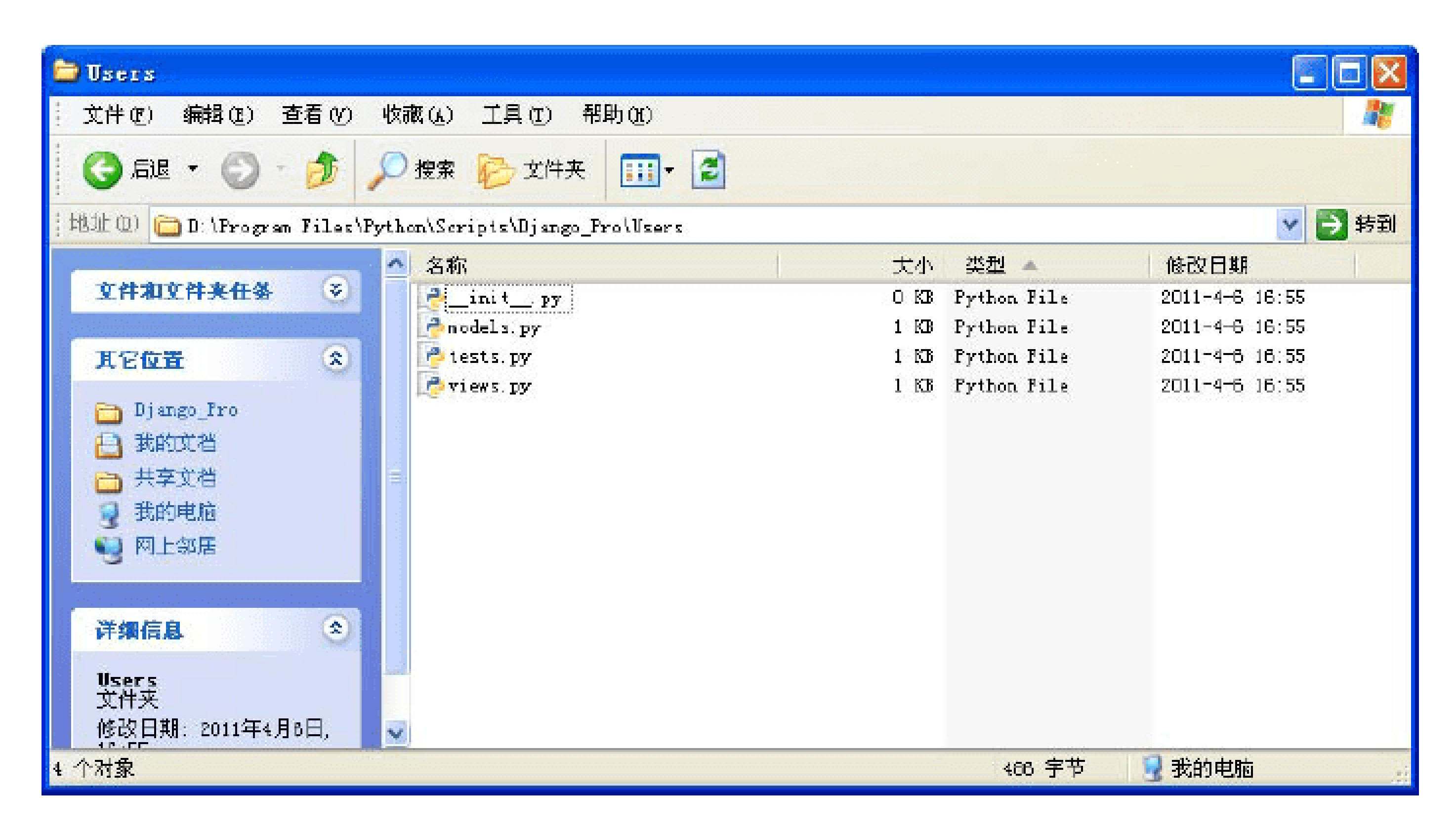


图 15-7　生成的Django应用目录

从图 15-7 可以看出，生成的 Users 目录下包含 4 个文件。

- __init__.py：空文件，但是必需的。用来将整个应用作为一个 Python 模块加载。
- models.py：定义数据模型相关的信息。
- tests.py：该应用的测试文件。
- views.py：包含与此模型的视图相关的信息。

15.4.5　基础知识——创建数据模型

创建 Django 应用后，需要定义保存在数据库中的数据。实际上，数据模型就是一组相关对象的定义，包括类、属性和对象之间的关系。为了创建数据模型，可以通过修改 Django 应用中的 models.py 文件来实现。此文件是一个 Python 脚本文件，其中定义了将要保存到数据库中的表。在下面的代码中，定义了一个 Users 表。

```
from django.db import models
# 创建一个 User 数据模型
class Users(models.Model):
    username = models.CharField('用户名',max_length=20)      #生成字段
    password = models.CharField('密码',max_length=20)
    realname = models.CharField('真实姓名',max_length=255)
    sex = models.CharField('性别',max_length=10)
    email = models.EmailField('电子邮箱',blank=True)

    def __str__ (self):
        return '%s'%(self.name)
```

在该段代码中，首先从 django.db 包中导入 models 模块，接着定义了一个名称为 Users 的类，该类继承自 models 中的 Model 类。在 Users 类的主体部分，定义了 5 个字段来描述用户的相关信息，包括用户名、密码、真实姓名、性别和电子邮箱。这里使用了 models 中的 CharField()函数来生成字段，在该函数中使用了两个参数：第一个参数表示在数据库中保存的字段名称，第二个参数表示该字段的最大长度限制。在该类的最后使用了__str__()方法来描述类。

创建了数据模型后，需要在 setting.py 文件中加入此应用。

```
INSTALLED_APPS = (
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Django_Pro.Users',
)
```



在 setting.py 文件中的 INSTALLED_APPS 元组中加入 Django_Pro.Users 值，用来将刚刚生成的 Django 应用加入到整个 Django 的项目中。

将此应用加入到项目中，可以继续使用 syncdb 在数据库中生成未创建的数据模型。在 Windows 命令中使用 cd 切换到 Django_Pro 目录，然后输入下面的内容。

```
manage.py syncdb
```

按回车键，输出如下：

```
Creating tables ...
Creating table Users_users
Installing custom SQL ...
Installing indexes ...
```

可以看出，这里生成了一个名为 Users_users 的表。通过 SQLite 工具查看 djangopro.db3 文件，可以看到其中已经生成了相应的数据表，并且有了对应的字段，如图 15-8 所示。

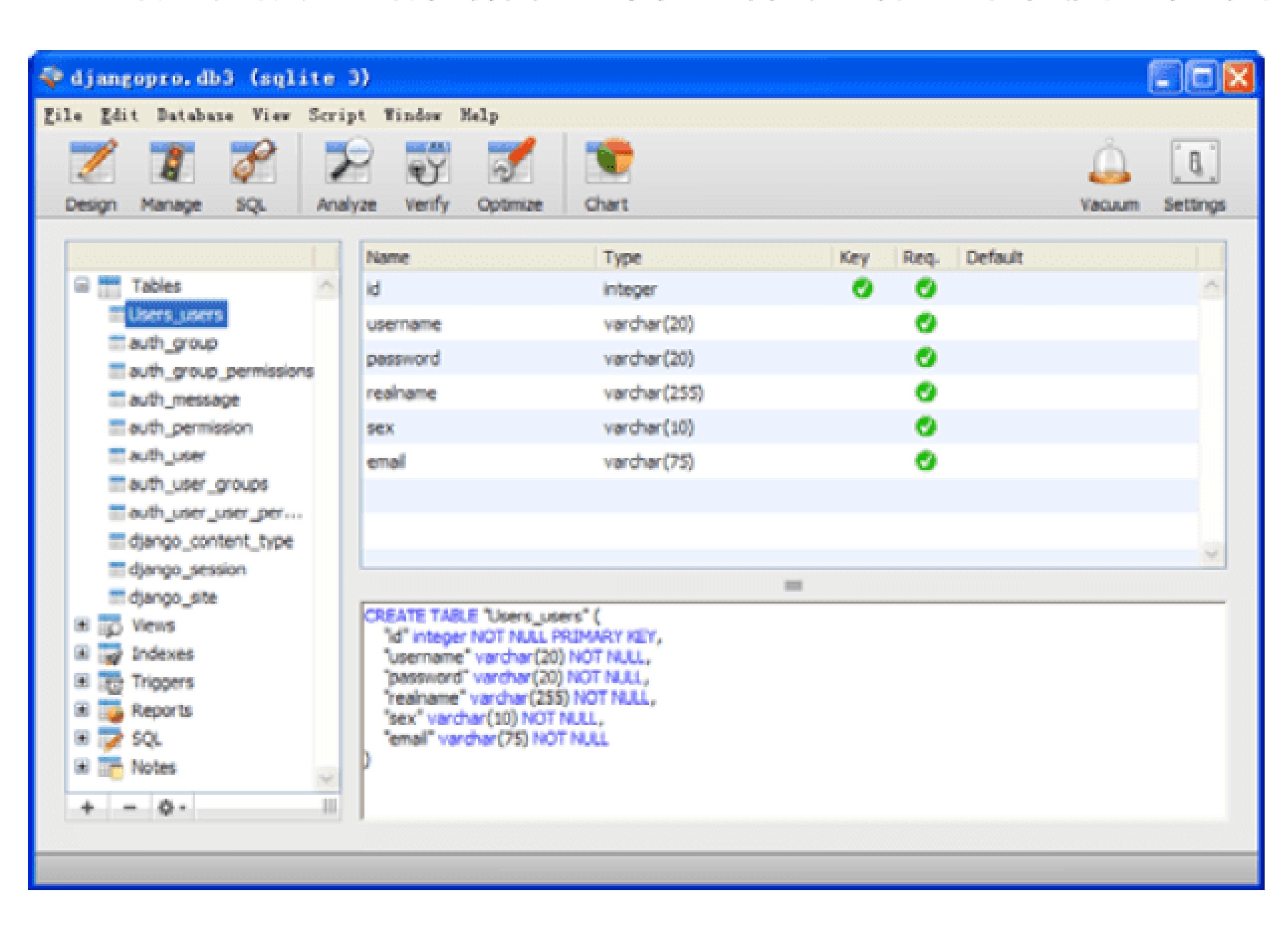


图 15-8　Users_users表

15.4.6　基础知识——URL设计

在 Django 项目中，URL 的设计是定义在配置文件 setting.py 的 ROOT_URLCONF 属性值中。在此属性值中定义了接收到 URL 后的处理方式，默认为根目录下的 urls.py 文件。

```
ROOT_URLCONF = 'Django_Pro.urls'
```



实际上，URL 的配置文件也是一个 Python 脚本文件，可以在此文件中设置访问的 URL。

当 Django 服务器启动后，对于接收到的每个 URL 请求，Django 会将此 URL 请求进行分解，得到相关的 URL 部分，并将此 URL 结果和 URL 配置文件中的设计进行匹配。在每次请求时，Django 的开发服务器将会打印此请求的相关信息。例如，在地址栏中输入 http://localhost:8000/Users/index.html，当此请求被发送到服务器时，Django 将会在 URL 的配置文件中匹配此请求 URL，并调用相应的方法。当服务器接收到此 URL 请求时，显示页面如图 15-9 所示。

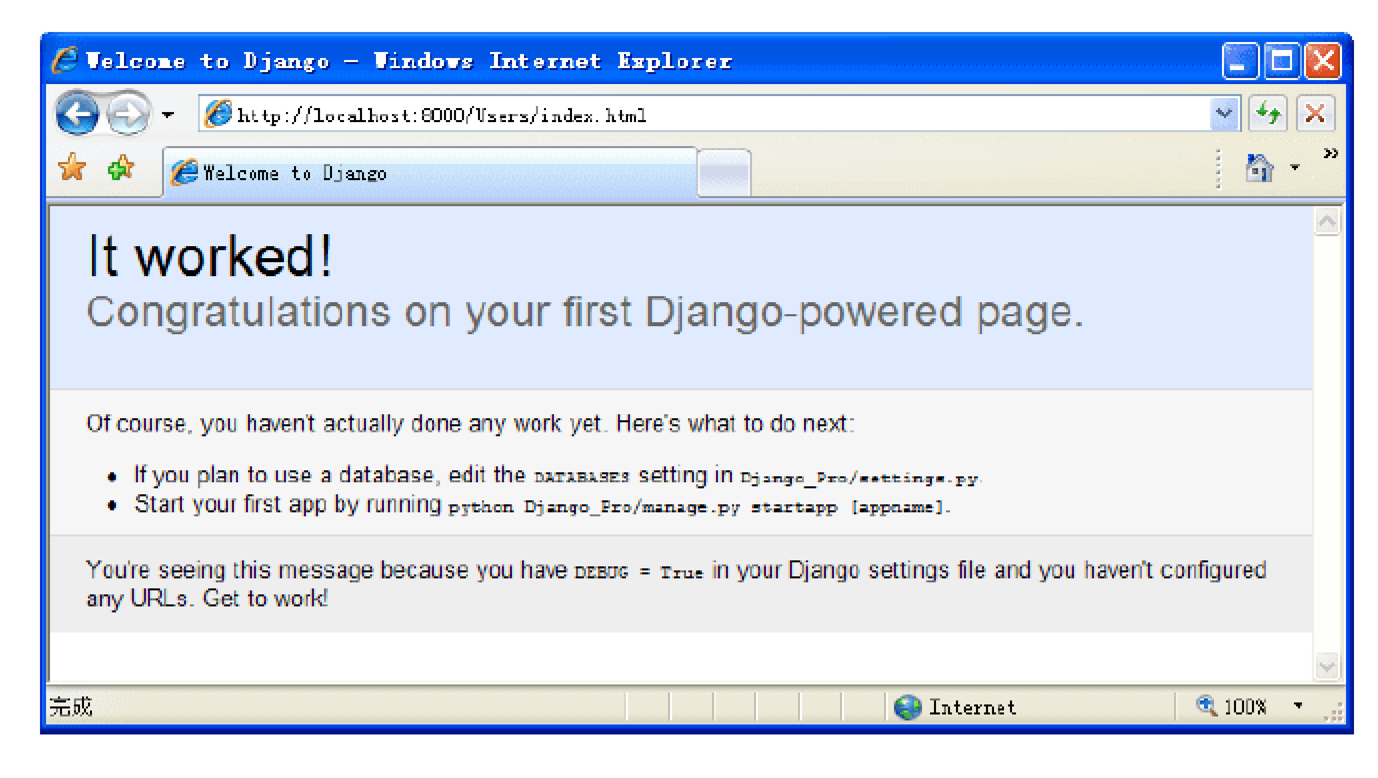


图 15-9　访问成功页面

同时在控制台中打印如下的 URL 请求信息：

```
[07/Apr/2011 09:02:25] "GET /Users/index.html HTTP/1.1" 200 2059
```

实际上，在根目录下的 urls.py 文件中，有两种方式来定义 URL：一种是直接定义特定的 URL 处理方式，另一种是递归使用子目录中的 URL 配置文件。下面就是这两种方式的配置代码。

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^$', 'Django_Pro.views.home', name='home'),
    (r'^Users/', include('Django_Pro.Users.urls')),
)
```

在 URL 的配置文件(urls.py)中，一般在文件的起始位置会使用语句 from django.conf.urls.defaults import patterns, include, url 将该文件代码所需要的类导入，此语句提供了 URL 的配置情况，例如 patterns 用来指定访问前缀模式。接下来有两个 URL 配置信息，每一项都会有两个部分：第一部分为 URL 的正则表达式匹配设置，第二部分为 URL 的处理函数。

这两个 URL 设计：第一个是直接使用 URL 前缀信息，其中^$表示访问 Web 服务器的根目录，Django_Pro.views.home 则表示此处理函数为根目录下的 views.py 文件中的 home()函数；在第二个 URL 匹配中，如果 URL 是以 Users/作为前缀，则将调用后面的处理函数，这个处理函数使用了 include 对象，用来表示具体的 URL 配置信息，可以从 Users 目录下的 urls.py 中查找。Users 目录下的 urls.py 文件的信息如下：

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^$', 'Users.views.index'),
    (r'^random_number/$', 'Users.views.random_number'),
)
```

当 Django 收到请求的 URL 地址后，会将处理后的 URL 针对其配置文件依次匹配。如果有匹配项，则调用相应的页面处理函数；如果没有与之相匹配的值，则会调用相应的错误信息。再次访问 http://localhost:8000/Users/index.html 地址，将会出现图 15-10 所示的结果。

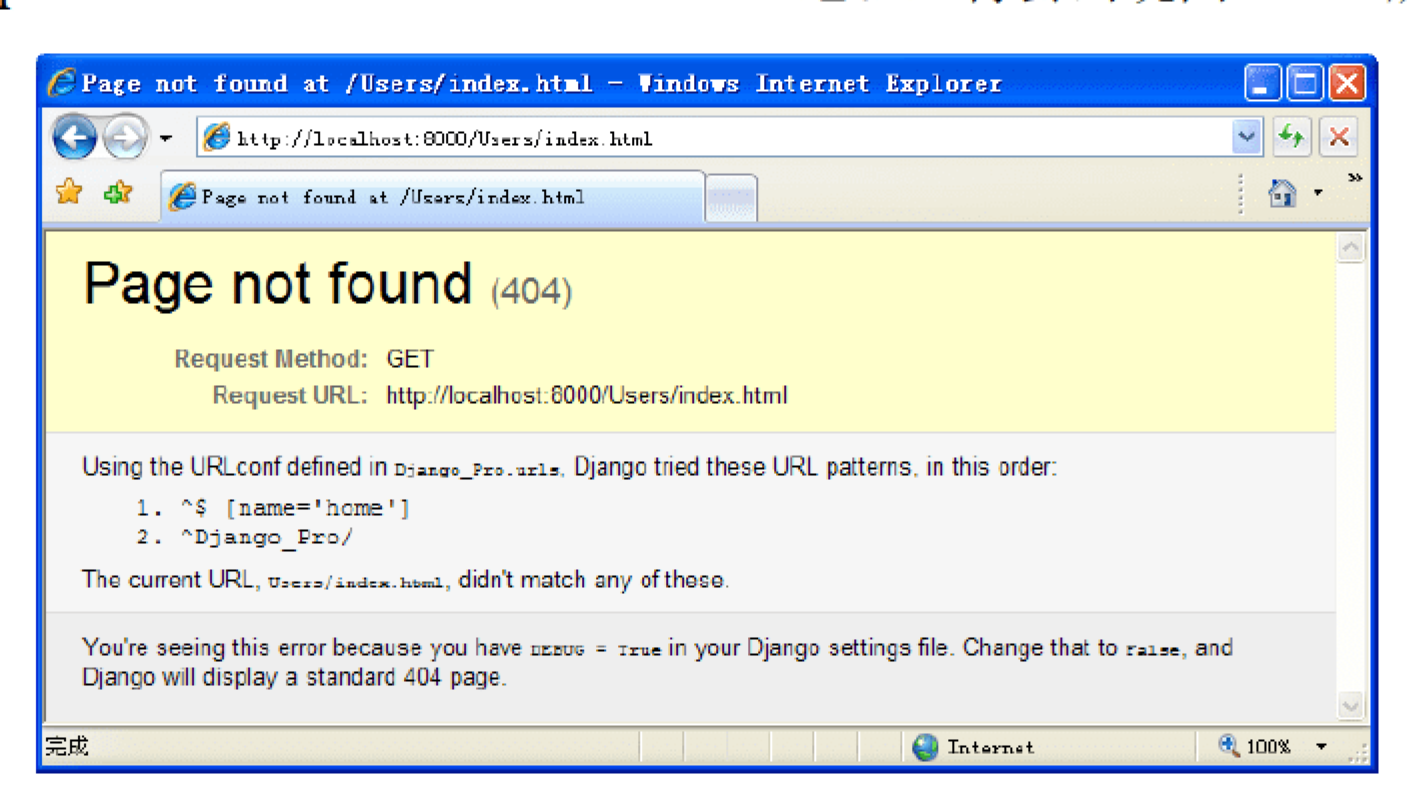


图 15-10　Django框架中的 404 错误页面

15.4.7　基础知识——创建视图

配置了 URL 之后，可以在视图文件中书写特定的视图函数(页面函数)。当 Django 服务器接收到特定的 URL 后，将会调用特定的视图函数。在 Django 中，视图分为动态视图和静态视图，下面作具体介绍。

1. 创建静态视图

静态视图即内容是固定不变的视图。下面在 Django_Pro 根目录下的 urls.py 文件中配置 Django_Pro.views.home 的视图函数，根目录下的 views.py 文件的代码内容如下：

```
from django.http import HttpResponse
def home (request):
    htmlStr='''
    <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>无标题文档</title>
        <style type="text/css">
        <!--
        body,td,th {
                    font-family: 华文行楷;
                    font-size: 24px;
       }
       -->
       </style></head>
       <body>
            欢迎进入窗内网的主页
       </body>
    </html>
    '''
    return HttpResponse(htmlStr)
```

在该段代码中，首先从 django.http 包中导入 HttpResponse 类。接着定义了一个名称为 home

的方法，此方法用于生成 HTTP 响应，其参数 request 为连接请求的对象，其中包含与请求相关的其他信息，如请求参数。例如，request.POST 可以用来表示使用 POST 方法请求的参数信息。在 home()方法的主体部分有两条语句：第一条语句定义了 htmlStr 变量，其中是 HTML 文档的内容；第二条语句使用 HttpResponse 类的构造函数生成了一个 HTTP 响应，其参数即为前面定义的 HTML 文档内容。

定义好页面函数，并配置了相应的 URL 后，打开浏览器，在地址栏输入 http://localhost:8000/，按回车键，出现图 15-11 所示的界面。

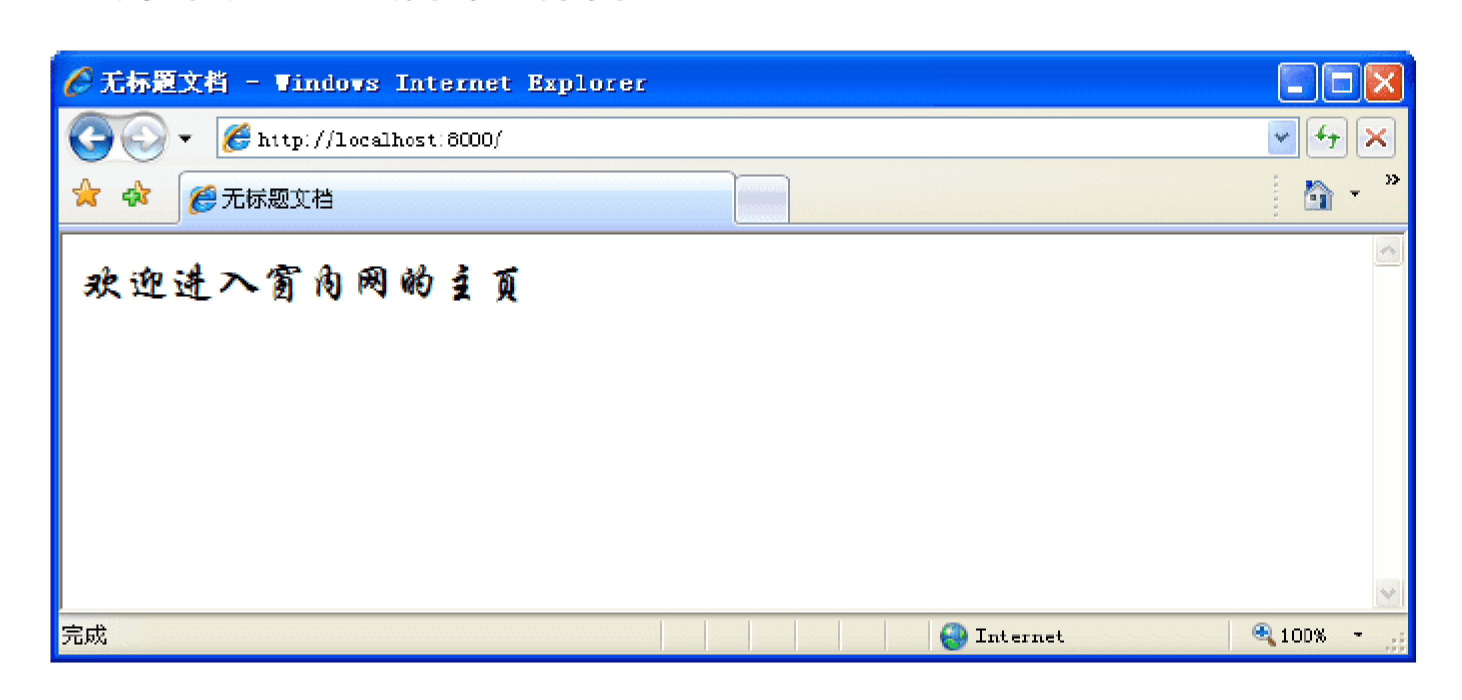


图 15-11　静态视图效果

2. 创建动态视图

在上面的例子中，仅仅显示了一个定义好的静态网页的内容，当然也可以显示动态信息。例如，在 Users 目录下的 views.py 文件中，使用 random_number()函数生成多个不同的随机数并显示出来。

```
from django.http import HttpResponse
import random
def random_number (request):                #定义 random_number()函数
    randNums = ''
    #进行 10 次循环，生成 10 个 10 以内的整数(可能重复)，并累加
    for num in range(10):
         randNums=randNums+str(random.randint(num,10-1))+'<br/>'
    htmlStr='''
    <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>无标题文档</title>
        <style type="text/css">
        <!--
        body,td,th {
                   font-family: 华文行楷;
                   font-size: 24px;
       }
       -->
       </style></head>
       <body>
           生成的随机数为<br/>%s'''%randNums+'''          #将生成的 10 个整数输出
       </body>
    </html>
    '''
    return HttpResponse(htmlStr)
```

在该段代码中，首先导入 random 模块，接着定义了位于 Users 目录下的 urls.py 文件中配置过的 random_number()方法。在该方法的主体部分，使用了 for ...in 循环 10 次，并在循环体内使用 random 模块的 randint()函数生成一个 10 以内的整数，同时采用叠加的方式将每次生成的随机数赋值给 randNums 变量。之后定义了 HTML 的内容，在 body 中输出生成的 10 个 10 以内的整数，最后返回一个 HttpResponse 对象。打开浏览器，在地址栏输入 http://localhost:8000/Users/random_number/，出现图 15-12 所示的效果。

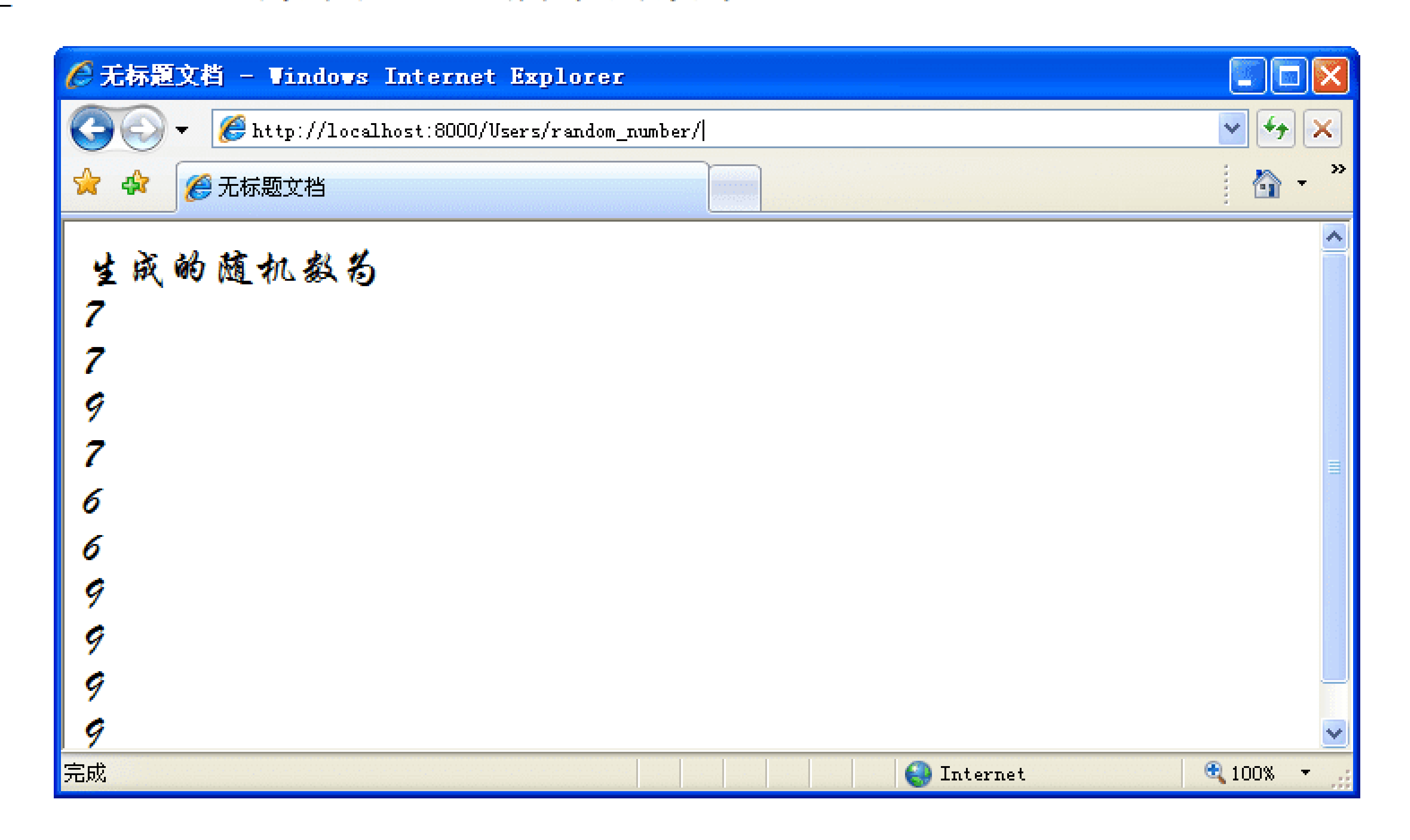


图 15-12　动态视图效果

15.4.8　基础知识——创建模板

通过上一节的介绍，大家发现视图与代码位于同一个文件中，即页面显示和数据没有分离。利用 Django 框架提供的模板系统，可以有效地分离页面显示和数据。另外，这种模板系统的文件可以重用，从而减少了代码的冗余及系统设计的复杂性。

Django 模板是利用{{variables}}和{%tags%}中嵌入的文本创建的。变量会使用它们表示的值进行计算和替换。标记用来实现基本的控制逻辑。模板可以用来生成任何基于文本的格式，包括 HTML、XML、CSV 和纯文本。

创建模板由下面几个步骤来完成。

(1) 在配置文件 setting.py 中，可以在 TEMPLATE_DIRS 属性中设置模板目录。

```
TEMPLATE_DIRS = (
    './templates',
)
```

在上面的设置中，将 settion.py 的当前目录 templates 作为模板文件的保存地址。



即使是在 Windows 系统平台下，在路径中也需要使用斜杠。

(2) Django 模板支持称为模板继承(Template Inheritance)的概念，它允许站点设计人员创建一个统一的外表，而不用替换每个模板的内容。可以通过使用块标记来定义骨干文档或基础文档并使用继承。这些块标记都使用一些包含内容的页面模板来填充。

配置模板目录后，在 templates 目录下新建 users 文件夹，并在 users 文件夹下新建 index.html 文件，该文件可以将显示分为两部分：一部分是显示，另一部分是代码。index.html 文件的内容如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta http-equiv="Content-Language" content="zh-cn" />
  <title> users index </title>
</head>
<body>
{% if latest_users_list %}
    <ul>
    {% for user in latest_users_list %}
        <li><a href="/users/{{ user.id }}/">{{ user.username }}</a></li>
    {% endfor %}
    </ul>
{% else %}
    <p>没有有效的数据！</p>
{% endif %}
</body>
</html>
```

(3) 打开 Django_Pro\Users\views.py 文件，然后在该文件中创建 index()函数。在 index()函数的主体部分，获取数据模型 Users 中的 username 列表，同时将获取的列表结果显示在 template\users\index.html 页面中，并使用 render_to_response()函数将该页面返回给客户端。views.py 文件的内容如下：

```
from django.shortcuts import render_to_response
from Django_Pro.Users.models import Users
def index(rq):
    #获取 Users 数据模型对应表中的数据
    latest_users_list = Users.objects.all().order_by('-username')[:5]
    return render_to_response('users/index.html',{
        'latest_users_list':latest_users_list
    })
```

(4) 配置请求的 URL 拦截路径。打开 Django_Pro 目录下的 URL 配置文件 urls.py，将 Django_Pro\Users\views.py 文件中的 index()函数所返回的 template\users\index.html 配置为 Django_Pro 项目的首页。修改 Django_Pro 目录下的 urls.py 文件内容如下：

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^$', 'Django_Pro.Users.views.index'),
)
```

(5) 此时 Users 数据模型 Users_users 表中的数据为空，因此需要编辑 SQL 语句，向该表中添加数据。

```
insert into Users_users(id,username,password,realname,sex,email)
values(1, 'maxianglin', 'maxianglin','马向林','女', 'maxianglin@mxl.com')
insert into Users_users(id,username,password,realname,sex,email)
values(2, 'wanglili', 'wanglili', '王丽丽','女', 'wanglili@wll.com')
insert into Users_users(id,username,password,realname,sex,email)
```

```
values(3,  'guoli',  'guoli', '郭力','男', 'guoli@gl.com')
```

到此为止，Django_Pro 项目的模板已经创建完毕。打开浏览器，输入 http://localhost:8000/，可以看到 Users 数据模型对应的 Users_users 表中的 username 列表，如图 15-13 所示。

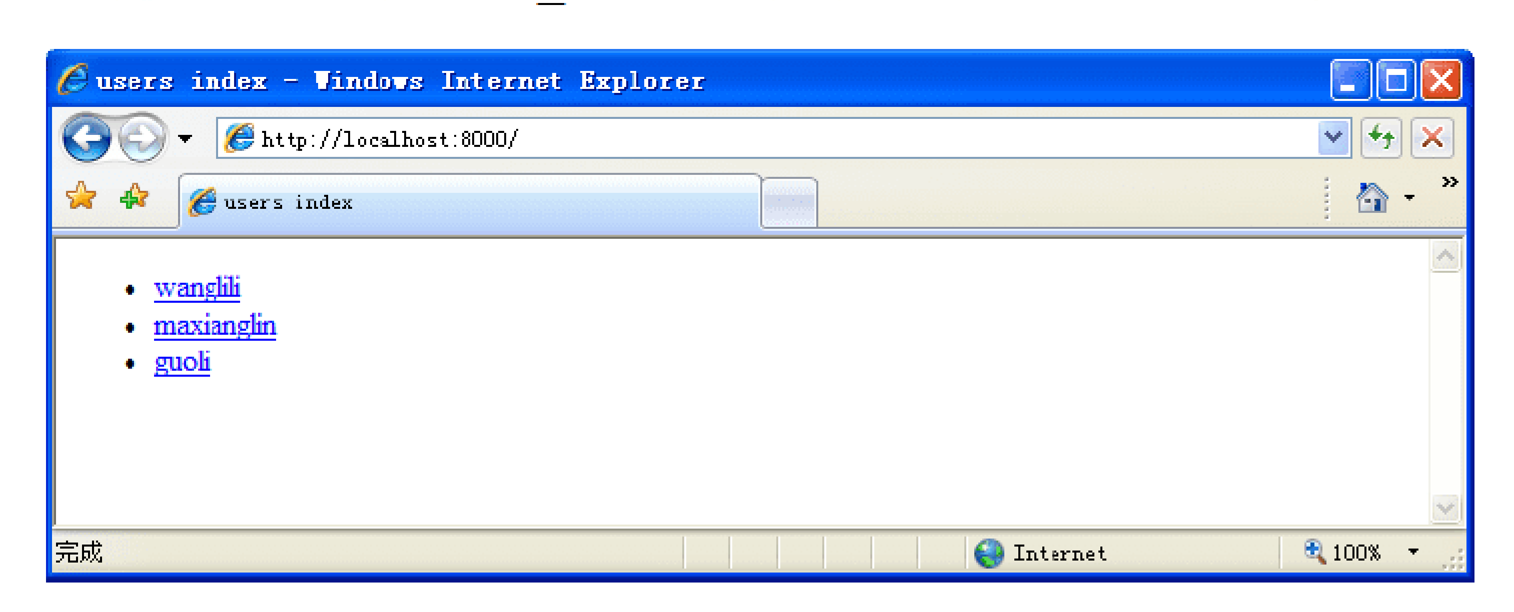


图 15-13　模板文件的显示

15.4.9　实例描述

在一起相处了 3 年的兄弟姐妹已经分隔了多年，回忆起学生时代的美好时光，真是感慨万千，有一种“少年不努力，老大徒伤悲”的感觉，但也有一种“每逢佳节倍思亲”的感慨，记得当年，班里的同学每人都有一本通讯录，帮助记忆其他同学的联系方式。我翻开通讯录的第一页，有“职业病”的我，刚学了 Django 框架，就想使用它来做一个通讯录。心动不如行动，下面就让我们一起来见证奇迹吧！

15.4.10　实例应用

【例 15-1】使用 Django 框架制作通讯录。

(1) 在 django-admin.py 文件的同级目录下新建 Address_Pro 项目。选择“开始”|“运行”，在“运行”对话框的文本框中输入 cmd，使用 cd 命令，切换至 Python 安装目录下的 Scripts 目录，该目录下有一个 django-admin.py 文件，然后在 Windows 命令中输入：

```
django-admin.py startproject Address_Pro
```

(2) 在 Address_Pro 目录下生成 Django 应用，该应用的名称为 Users。继续在 Windows 命令中使用 cd 命令，切换至 Address_Pro 所在的路径下，接着输入：

```
manage.py startapp Users
```

(3) 在 Address_Pro 目录下新建 template 文件夹，该目录作为 Address_Pro 的模板目录。

(4) 打开 Address_Pro 目录下的 settings.py 文件，修改 DATABASES 字典如下：

```
DATABASES = {
    'default': {
        'ENGINE': 'sqlite3', # Add 'postgresql_psycopg2', 'postgresql', 'mysql',
'sqlite3' or 'oracle'.
        'NAME': './addresspro.db3',  # Or path to database file if using sqlite3
        'USER': '',                  # Not used with sqlite3
        'PASSWORD': '',              # Not used with sqlite3
        'HOST': '',        # Set to empty string for localhost. Not used with sqlite3
```

```
        'PORT': '',       # Set to empty string for default. Not used with sqlite3
    }
}
```

(5) 指定模板目录。修改 settings.py 文件中的 TEMPLATE_DIRS 元组为：

```
TEMPLATE_DIRS = (
    './template',
)
```

(6) 打开 Address_Pro\Users 目录下的 models.py 文件，新建 User 实体类，使之生成 Users_user 的数据模型，该数据模型中有 4 个字段，分别记录了同学的姓名(pname)、性别(sex)、年龄(age)和家庭住址(addr)，然后使用__str__()方法来描述 User 类。models.py 文件的内容如下：

```
#encoding=utf-8
from django.db import models
class User(models.Model):
    # 姓名
    pname = models.CharField(max_length=150)
    # 性别
    sex = models.CharField(max_length=10)
    # 年龄
    age = models.IntegerField(blank=True)
    # 家庭住址
    addr = models.CharField(max_length=255)
    def __str__ (self):
        return '%s'%(self.name)
```

(7) 修改 Address_Pro 目录下 settings.py 文件中的 INSTALLED_APPS 元组值，在该元组中添加 Users 应用，从而生成所对应的数据模型 Users_user。

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'Address_Pro.Users',
)
```

(8) 创建数据模型。再次打开 Windows 命令，使用 cd 命令将目录切换至 Address_Pro 目录下，然后输入：

```
manage.py syncdb
```

这时会出现下面的内容，表示创建表成功。

```
Creating tables ...
Creating table auth_permission
Creating table auth_group_permissions
Creating table auth_group
Creating table auth_user_user_permissions
Creating table auth_user_groups
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
```

```
Creating table Users_users
```

(9) 在 Address_Pro 目录下的 Users 目录下新建 add.py 文件，该文件用于向通讯录列表中添加数据。为简单起见，这里没有操作数据库，只是使用列表保存了 7 条数据，然后将该列表传递到模板页。add.py 文件的内容如下：

```
# -*- coding: utf-8   -*-
from django.shortcuts import render_to_response
address = [
  {'name':'alan', 'sex':'男','age':'25','address':'河南省郑州市'},
  {'name':'阿汐', 'sex':'男','age':'21','address':'河南省郑州市'},
  {'name':'sgicer', 'sex':'男','age':'23','address':'河南省郑州市'},
  {'name':'tidewind', 'sex':'男','age':'32','address':'河南省安阳市'},
  {'name':'cood', 'sex':'男','age':'22','address':'河南省安阳市'},
  {'name':'北极乞丐', 'sex':'男','age':'25','address':'河南省郑州市'},
  {'name':'北斗', 'sex':'男','age':'15','address':'河南省安阳市'}
       ]
def index(request):
       return render_to_response('list.html',{'address': address})
```

(10) 制作模板页 list.html。在 template 目录下新建 list.html 页面作为模板页，在该页面中获取 add.py 文件传递过来的 address 集合，并循环遍历输出集合中的元素。同时在该页面中编辑 JavaScript 脚本，达到鼠标移动到某一行时该行背景加深显示的效果。list.html 页面代码如下：

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>无标题文档</title>
<style type="text/css">
table{border:1px solid #003366;border-width:1px 0 0 1px;margin:2px 0 2px
0;text-align:center;border-collapse:collapse;}
   td,th{border:1px solid #003366;border-width:0 1px 1px 0;margin:2px 0 2px
0;text-align:center;
   background-color: #FFFFFF;}
   th{text-align:center;font-weight:600;font-size:14px;background-color:
#003366; color:#FFFFFF};
   tr.t1 td {background-color:#fff;}/* 第一行的背景色 */
   tr.t2 td {background-color:#eee;}/* 第二行的背景色 */
   tr.t3 td {background-color:#ccc;}/* 鼠标经过时的背景色 */
</style>
</head>
<body>
<h2>通讯录</h2>
<table width="100%" cellpadding="0" cellspacing="0" id="tab" >
  <tr>
   <th width="17%">姓名</th>
   <th width="17%">性别</th>
   <th width="22%">年龄</th>
   <th width="44%">地址</th>
  </tr>
  {% for user in address %}
  <tr>
  <td>{{ user.name }}</td>
  <td>{{ user.sex }}</td>
```

```
  <td>{{ user.age }}</td>
  <td>{{ user.address }}</td>
  </tr>
{% endfor %}
</table>
<script type="text/javascript">
<!--
    var Ptr=document.getElementById("tab").getElementsByTagName("tr");
  function $() {
      for (i=1;i<Ptr.length+1;i++) {
       Ptr[i-1].className = (i%2>0)?"t1":"t2";
      }
  }
  window.onload=$;
  for(var i=0;i<Ptr.length;i++) {
       Ptr[i].onmouseover=function(){
        this.tmpClass=this.className;
       this.className = "t3";
     };
       Ptr[i].onmouseout=function(){
      this.className=this.tmpClass;
     };
}
//-->
</script>
</body>
</html>
```

(11) 打开 Address_Pro 目录下的 urls.py 文件，修改匹配的 URL。urls.py 文件内容如下：

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^add/$', 'Users.add.index'),
)
```

15.4.11 运行结果

打开 IE 浏览器，在地址栏输入 http://localhost:8000/add/，按回车键，跳转至 list.html 页面，在该页面中显示了通讯录列表。当鼠标移动到某一行时，背景颜色改变。运行结果如图 15-14 所示。

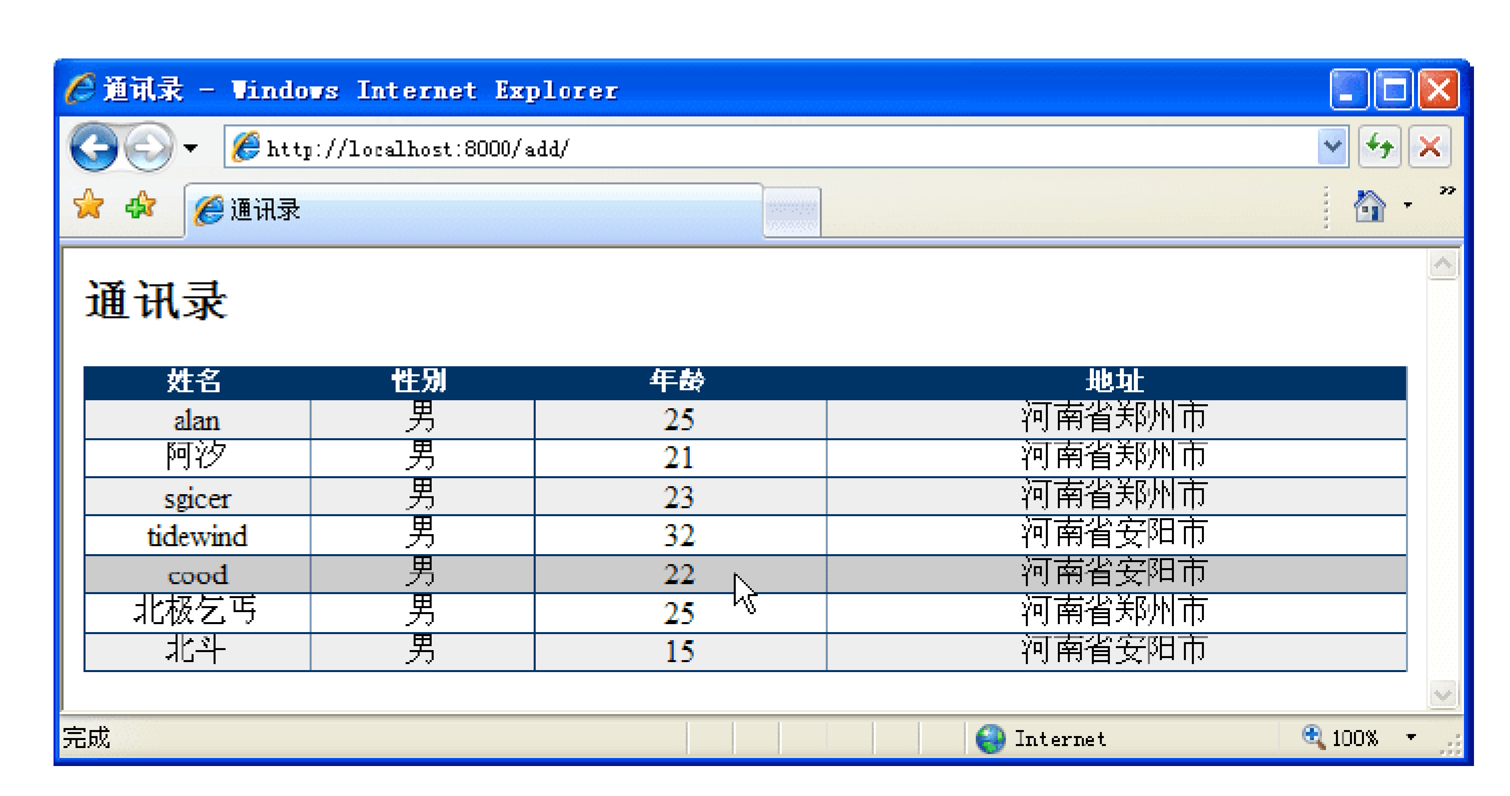


图 15-14　通讯录列表

15.4.12　实例分析



源码解析

在上面的例子中，在 Address_Pro 目录下的 urls.py 文件中配置的匹配 URL 为(r'^add/$', 'Users.add.index')，表示当访问 add 时，程序将执行 Users 应用中的 add.py 文件中的 index()方法，即转向 list.html 页面，并将 add.py 文件中的 address 列表作为参数传递给 list.html 页面(address 是一个列表类型的集合，集中的每个元素又是一个含有 4 个元素的字典)。在 ist.html 页面中使用 for ...in 循环遍历 address 集合，并将集合中的数据输出。

15.5 使用Django框架的Session实现购物车

通过上一节的讲述，大家已经了解了 Django 框架的基本应用，如果只是构建一个简单的 Web 应用，这些已经足够了。为了能满足用户的需求，Django 框架还提供了高级功能，用来构建丰富的 Web 应用。

 **视频教学：** 光盘/videos/15/ Diango 框架的高级应用.avi　　长度：11 分钟

15.5.1 基础知识——界面管理

Django 框架的最大特点之一就是内置了一个很好的管理界面，可以在该管理界面中对项目的数据进行管理，包括添加数据和删除数据等功能。下面来演示 Django 框架的管理界面。

(1) 在 15.4 节 Django_Pro 项目的配置文件 settings.py 中，修改 INSTALLED_APPS 元组变量值，加入 django.contrib.admin 元素值。

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'django.contrib.admin',
    'Django_Pro.Users',
)
```

(2) 打开 Windows 命令，使用 cd 命令，切换至 Django_Pro 目录下，然后调用 manage.py 文件的 syncdb 子命令，在数据库中创建 django.contrib.admin 所对应的表。

```
manage.py syncdb
Creating tables ...
Creating table django_admin_log
Installing custom SQL ...
Installing indexes ...
```

从输出信息可以看出，已生成了管理界面所需的表 django_admin_log。如果之前没有生成管理员用户，则在此处可能需要输入管理员用户的用户名和密码。由于在 15.4 节创建表时已经创建了用户名和密码，则此处不再提示输入。

(3) 修改 Django_Pro 目录下的 urls.py 文件中的必要部分来增加对管理界面的支持，下面是修改后的 urls.py 文件。

```
from django.conf.urls.defaults import patterns, include, url/
from django.contrib import admin
admin.autodiscover()
urlpatterns = patterns('',
    (r'^$', 'Django_Pro.views.home', name='home'),
    (r'^Users/',include('Django_Pro.Users.urls')),
    (r'^admin/',include(admin.site.urls)),
```

```
)
```

在该文件中，首先从 django.contrib 导入了 admin，并调用其 autodisconver()方法，接着在 patterns()方法中设置了 URL 的匹配数据。



由于本书使用的 Django 版本为 1.3，所以在配置以 admin/的访问路径时，需要设置为 include(admin.site.urls)。如果使用的 Django 为 1.3 以下的版本，则设置为 admin.site.root 即可。

(4) 在 Users 目录下生成一个 admin.py 文件，其内容如下：

```
from django.contrib import admin
from models import Users
admin.site.register(Users)
```

在该段代码中，使用 admin.site 模块的 register()方法将此数据模型关联到管理界面中。

打开 IE 浏览器，在地址栏输入 http://localhost:8000/admin/，按 Enter 键，将会出现如图 15-15 所示的页面。

如果需要将 Django 的管理界面汉化，则需要修改 settings.py 文件中的 LANGUAGE_CODE 属性值，修改后的结果如下：

```
LANGUAGE_CODE = 'zh-CN'
```

再次访问 http://localhost:8000/admin，汉化后的管理员登录界面如图 15-16 所示。在这里可以使用前面创建的管理员用户名和密码登录，登录后的界面如图 15-17 所示。

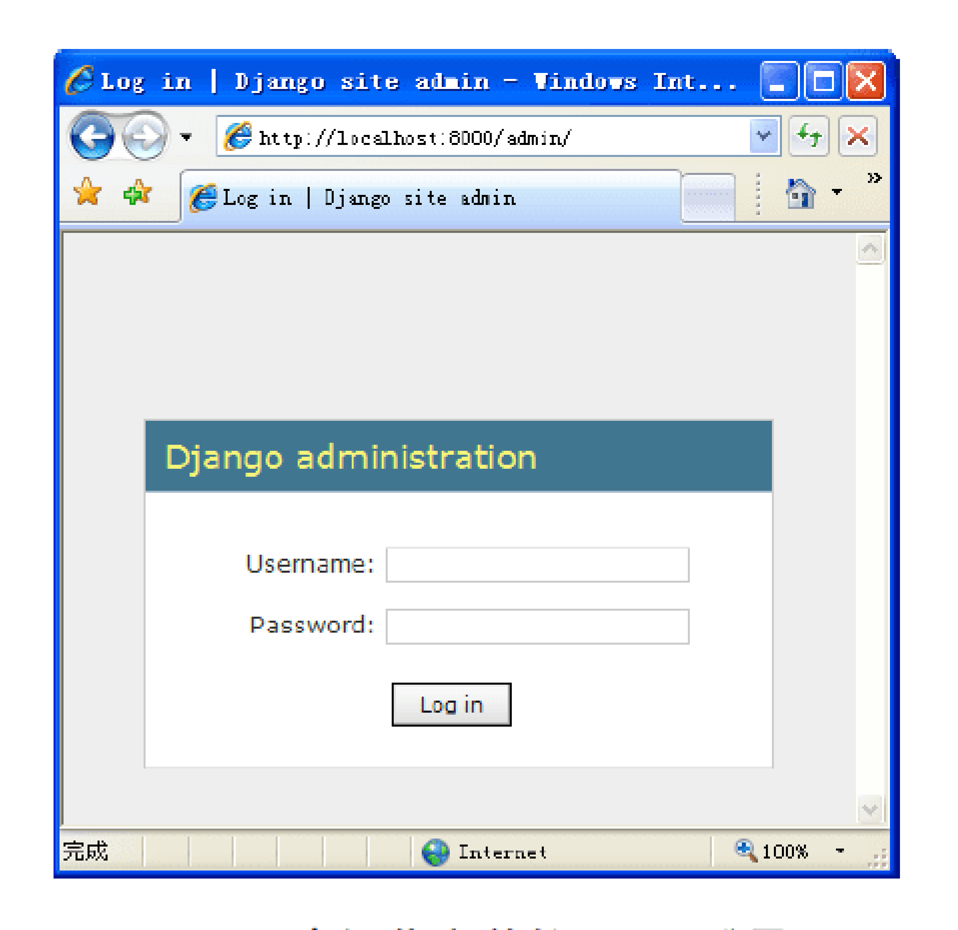


图 15-15　未汉化之前的admin登录界面



图 15-16　汉化后的admin登录界面

从图 15-17 可以看出，其中包含了已经定义过的数据模型 Users(对应的表为 Users_users)，可以通过此管理界面来增加和删除数据。另外，这里还有 Django 框架内置的几个数据模型：用户数据模型(对应的表为 auth_user)、组数据模型(对应的表为 auth_group)以及站点数据模型(对应的表为 django_site)。Django 框架中的管理员用户和密码就保存在用户数据模型，即 auth_user 表中。



图 15-17　Django管理界面内容

15.5.2　基础知识——生成数据表数据

在 15.5.1 节中，已经成功登录到管理界面，从图 15-17 可以看到在每个数据模型之后都有两个超链接，即增加和修改。下面分别演示这两种不同的操作，从而插入/修改数据库中的数据。

1．使用Django框架的管理界面向数据表中插入数据

Django 自带的后台管理界面可以实现向数据表中插入数据，下面以向 Users_users 表中增加记录为例来演示 Django 如何向数据表中插入数据。

(1) 单击 Users 域中的 Userss 之后的“增加”超链接，页面跳转至 admin/Users/users/add/ 路径下的视图，在此视图中输入用户信息，如图 15-18 所示。

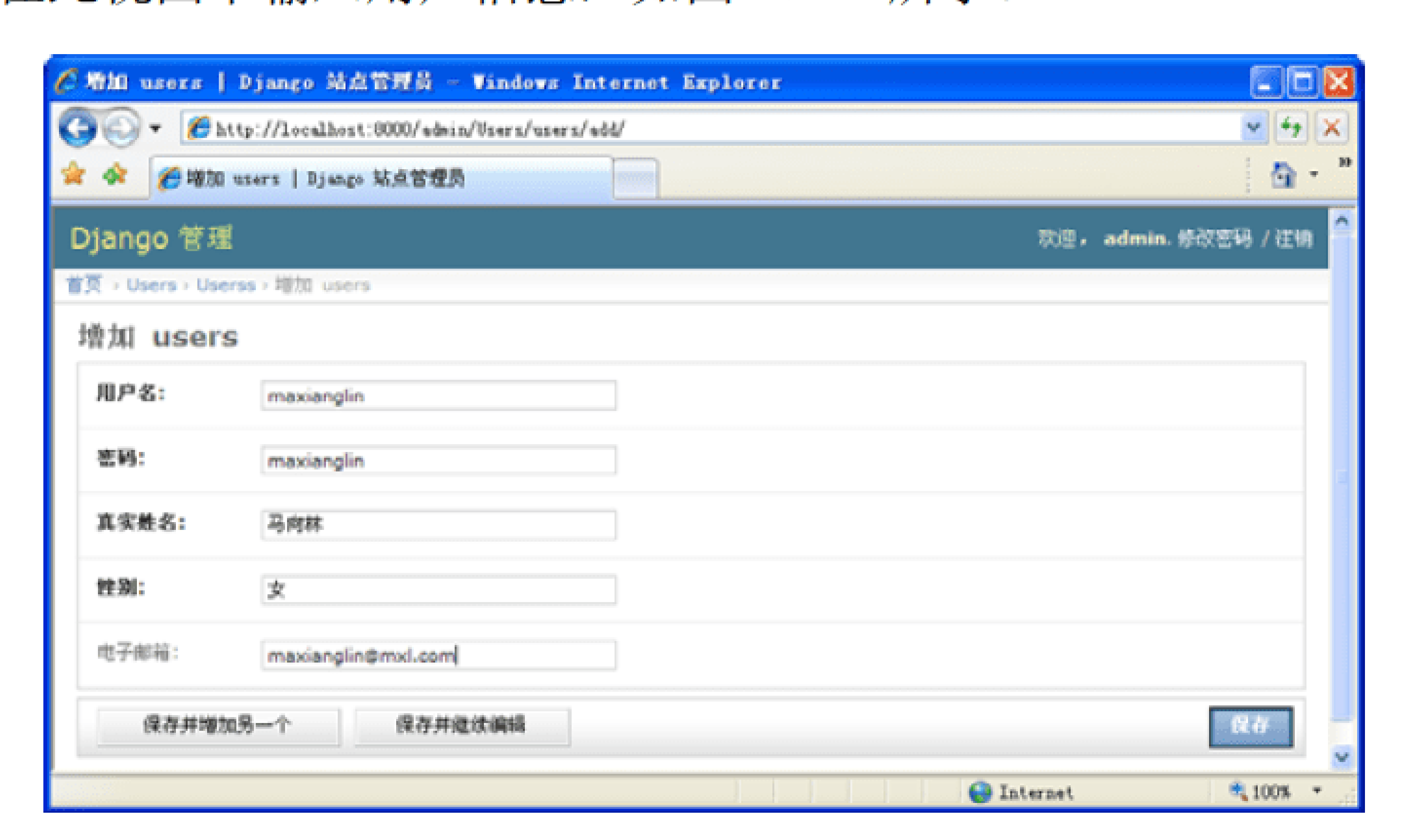


图 15-18　增加users界面

(2) 当输入完整用户信息之后，单击“保存并增加另一个”按钮，页面跳转至 admin/Users/users/视图，提示用户“添加成功”，如图 15-19 所示。

从图 15-19 可以看出，Users 数据模型中只有一条记录，显示的是用户的用户名。这里需要修改 Users 目录下 models.py 文件中的 Users 类，修改后的代码如下：

```
class Users(models.Model):
    # 省略类中定义的 username、password、realname、sex 和 email 属性
    def __str__(self):
        return '%s'%(self.username)
```



图 15-19　成功向数据表中增加数据

(3) 测试是否添加成功。测试的方法很多，最直观的方法就是查看数据表中是否存在该条记录。当然也可以单击该条数据的用户名，查看是不是刚增加的数据。单击用户名为 maxianglin 的数据，显示该条数据的详细信息，如图 15-20 所示。

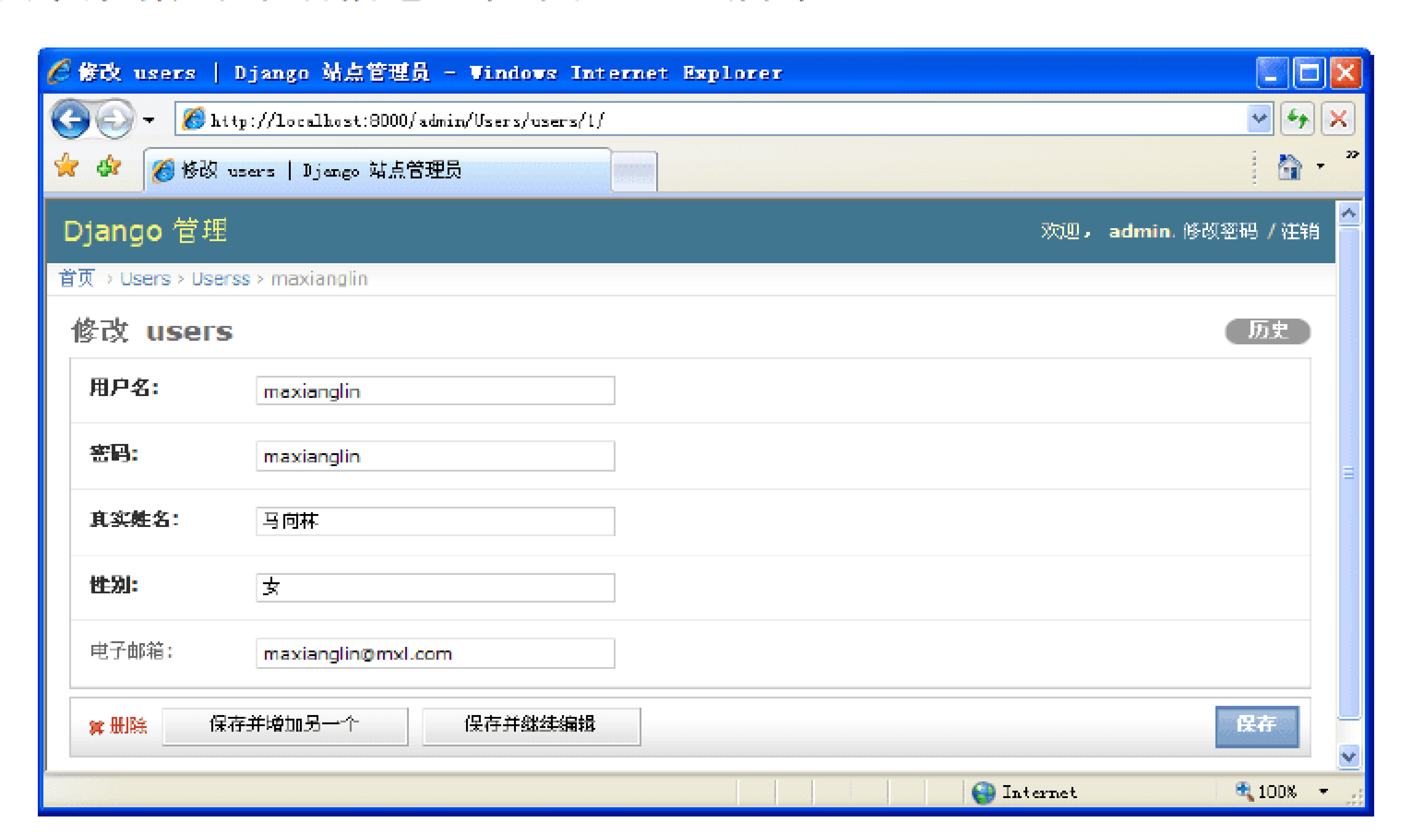


图 15-20　用户的详细信息

2．使用Django框架的管理界面修改数据表中的数据

通过上面的演示，可以发现 Django 自带的后台管理系统是非常强大的，无需开发者做任何配置，就可以实现向数据表中增加数据的功能。其实，Django 框架的管理界面不仅可以实现增加数据功能，也能实现修改、删除和查询功能。由于删除和查询非常简单，这里不再叙述，下面来演示如何通过 Django 框架的管理界面实现修改数据表中的数据。

(1) 单击后台管理系统首页中的 Users 域中的 Users 超链接，页面跳转至“选择 users 来修改”界面，界面效果如图 15-19 所示。

(2) 单击用户列表，页面跳转至“修改 users”管理界面，同时显示用户的详细信息。修改后的用户信息如图 15-21 所示。

(3) 单击“保存并增加另一个”按钮，系统提示“修改成功”信息，如图 15-22 所示。

(4) 测试是否修改成功。单击用户名为 mxl050221 的超链接，显示该用户的详细信息，如图 15-23 所示。



图 15-21 修改后的用户信息



图 15-22 修改数据成功



图 15-23 用户的详细信息

3．使用Django提供的API生成数据表数据

上面讲述的两种操作数据表的方式都采用了 Django 框架的管理界面。其实，Django 框架

还提供了生成数据模型数据的另一种方法，这种方法简单和直观。该方法使用 Django 提供的 API 来生成数据，即直接调用数据模型的构造函数来生成一个对象，然后使用该对象的 save()方法将生成的对象保存在数据库中。使用类的 objects 对象的 all()方法可以获取该数据模型中的所有数据，但是此方法需要使用 manage.py 文件中的 shell 子命令来生成一个控制器才得以实现。下面的代码完成了向数据表中添加一条记录的功能。

```
#在 Windows 命令中使用 cd 命令转到 Django_Pro 目录下，然后输入 manage.py shell
manage.py shell
Python 2.5.4 (r254:67916, Dec 23 2008, 15:10:54) [MSC v.1310 32 bit (Intel)]
on
win32
Type "help", "copyright", "credits" or "license" for more information.
# 下面为代码部分
>>> from Users.models import Users          #导入 Users 类
>>>
users=Users(username='yinguopeng',password='yinguopeng',realname='Youn',sex=
'male',email='yinguopeng@ygp.com')          # 创建 Users 类的构造函数生成 Users 对象
>>> users.save()                            # 使用 save()方法向数据表中插入数据
>>> usersList=Users.objects.all()           # 获取 Users 数据模型中的所有数据
>>> print usersList                         # 输出获取的所有数据
[<Users: mxl050221>, <Users: yinguopeng>]   # 输出的结果
```

在此段代码中，首先使用 manage.py 中的 shell 子命令生成了一个新的控制台，然后从 Users 应用中导入 Users 类，并使用其构造函数生成一个 Users 对象，其中的参数即为各个属性的内容。接着调用了 Users 对象的 save()方法将生成的 users 对象保存到数据表中，并使用 all()方法获取 Users 数据模型中的数据列表。最后使用 print 方法输出此列表信息。可以看出，目前数据表 Users_users 中已经有了两条记录，其中一条是使用管理界面创建的数据，而另外一条则是刚刚使用 API 生成的。

15.5.3 基础知识——Session的应用

Django 完全支持 Session，Session 的实现可以根据服务器来决定，一般保存在 Cookie 中，通过此值可以判断不同的连接并与之交换数据。在 Django 框架中，Session 将保存在 request 对象的 session 值中，此值是一个字典对象，可以通过字典的相关操作来改变 HTTP 的 session 值。

1. 启用Session

如果需要在 Django 项目中启用 Session 功能，则需要修改 settings.py 文件中的 MIDDLEWARE_CLASSES 元组类型的属性值，在该元组中加入下面语句。

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

同时在此配置文件中的 INSTALLED_APPS 元素类型的属性值中加入下面语句。

```
'django.contrib.sessions',
```

如果此前没有生成相关的数据表，则需要调用 manage.py 文件中的 syncdb 子命令来创建相应的表。到此为止，就成功启用了 Django 框架中的 Session。



在实际应用中，还需要在浏览器中开启 Cookie 功能。

2. 创建Session并保存用户数据

下面使用最常用的登录页面为例具体介绍如何使用 Session 对象来保存用户输入的数据。

首先在 templates 目录下新建 login.html 文件，该文件为程序的登录页面。在该页面中有一个用户名文本框和一个密码框。login.html 文件的代码如下：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312" />
<title>后台登录</title>
<style type="text/css">
<!--
body {
    margin-left: 0px;
    margin-top: 0px;
    margin-right: 0px;
    margin-bottom: 0px;
    overflow:hidden;
}
.STYLE3 {color: #528311; font-size: 12px; }
.STYLE4 {
    color: #42870a;
    font-size: 12px;
}
-->
</style>
</head>
<body>
  {% if not username %}       <!--如果用户名为空，显示文本框供用户输入-->
    <form method="GET" action="/login/">
      <table width="100%" border="0" cellspacing="0" cellpadding="0">
        <tr>
          <td  width='40%' height="30"><div align="right"><span
class="STYLE3">用户名</span></div></td>
          <td width='60%' height="30" align='left'><input type="text"
name="username"  style="height:18px; width:130px; border:solid 1px #cadcb2;
font-size:12px; color:#81b432;"></td>
        </tr>
        <tr>
            <td height="30"><div align="right"><span class="STYLE3">密码
</span></div></td>
            <td height="30" align='left'><input type="password"
name="password"  style="height:18px; width:130px; border:solid 1px #cadcb2;
font-size:12px; color:#81b432;"></td>
        </tr>
           <tr>
            <td height="30"> </td>
            <td height="30"><input type="submit" value="登录"/></td>
       </tr>
      </table>
     </form>
  {% else %}     <!--如果获取的用户名不为空，则表示已经登录过，提示用户登录成功信息-->
    欢迎您：{{username}}！您已成功登录！
```

```
  {% endif %}   <!--结束标记-->
</body>
</html>
```

在该页面中使用了 Django 的 if 模板标签，根据获取的 Session 对象中的 username 为条件显示不同的内容：当获取的 username 为空时，则显示两个文本框和一个提交按钮的表单域；当获取的 username 不为空，则提示用户“登录成功”信息。其他的 HTML 内容和普通的登录界面内容相同，这里不做过多的描述。

接下来在 Django_Pro 目录下的 views.py 文件中导入需要的类，代码如下：

```
from django.shortcuts import render_to_response
from Django_Pro.Users.models import Users
from django.http import HttpResponseRedirect
```

最后在 Django_Pro 目录下的 views.py 文件中编辑 login()方法，用来处理登录操作。在该方法中首先需要获取用户输入的用户名和密码，然后调用数据模型中的 all()方法获取数据表中的数据，并循环输出数据列表，获取每条数据记录，使之与获取的用户名和密码做比较，检查登录用户是不是合法用户。最后返回 login.html 页面。

```
def login (request):
    # 获取用户输入的用户名
    username=request.GET.get('username',None)
    # 获取用户输入的密码
    password=request.GET.get('password',None)
    # 如果用户输入的用户名不为 None
    if username is not None:
        # 调用数据模型类的 objects.all()方法获取 Users_users 数据表的数据列表
        usersList=Users.objects.all()
        # 循环遍历数据列表
        for users in usersList:
            # 判断用户输入的用户名和密码是否在数据表中存在
            if users.username == username and users.password == password:
                # 将该用户输入的用户名保存至 Session 对象中
                request.session['username']  = username
                # 返回 login.html 页面，并将 username 的值传递过去
                return render to response('login.html',{'username':username})
    return render_to_response('login.html')
```

在该段代码中定义了一个 login()方法，该方法用于处理用户的登录。在该方法的主体部分，首先使用 request.GET.get()方法获取通过 GET 请求发送过来的 username 和 password 的值，然后判断传递过来 username 是否为 None，如果不为 None，则从数据表 Users_users 中获取所有的数据。对于数据集合中的每条记录，判断数据表中的用户名和密码是否和从 HTTP 请求中传递过来的数据相等。如果两者相等，则返回登录界面，显示“登录成功”信息。

3. 配置URL匹配信息

通过上面的步骤，已经成功启用 Session，并将用户输入的数据保存至 Session 中。那么在浏览器中如何查看登录页面(login.html)的内容呢？下面就来配置一下 urls.py 文件。

修改 Django_Pro 目录下的 urls.py 文件，加入下面的 URL 匹配信息。

```
(r'^login/$','Django_Pro.views.login'),
```

4．测试结果

在浏览器的地址栏输入 http://localhost:8000/admin，出现登录页面，如图 15-24 所示。在此页面中输入用户名和密码后，单击“登录”按钮，将触发定义在 Django_Pro 目录下的 views.py 文件中的 login()方法，获取用户输入的用户名和密码。在处理函数中，Django 将会根据 GET 的值进行处理。当 username 为空或者密码不匹配时，则返回登录页面，如果匹配，则会显示图 15-25 所示的页面。

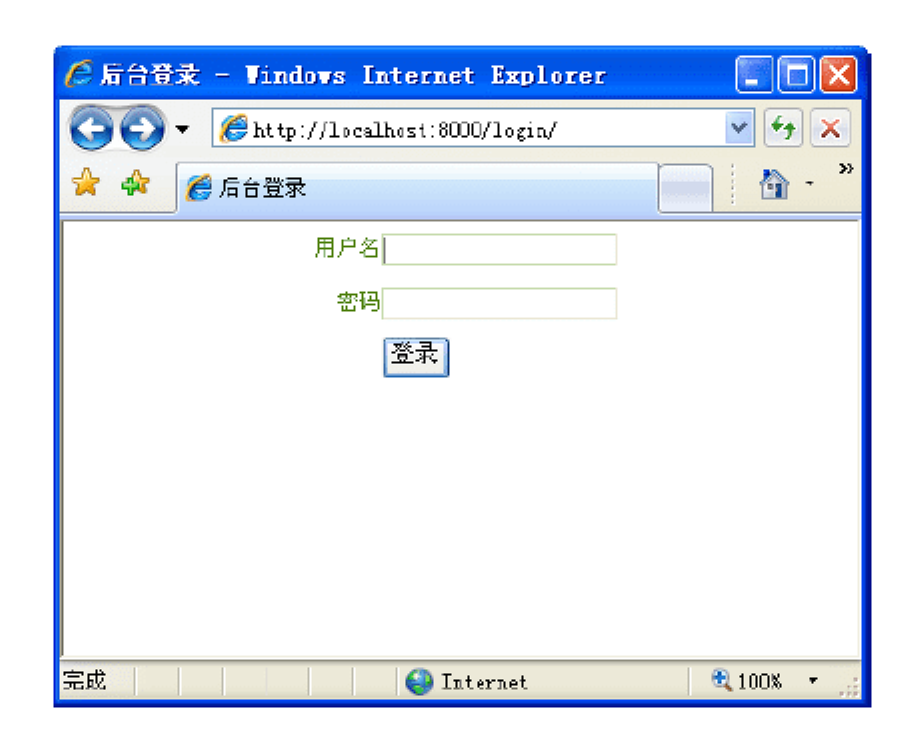


图 15-24　登录页面

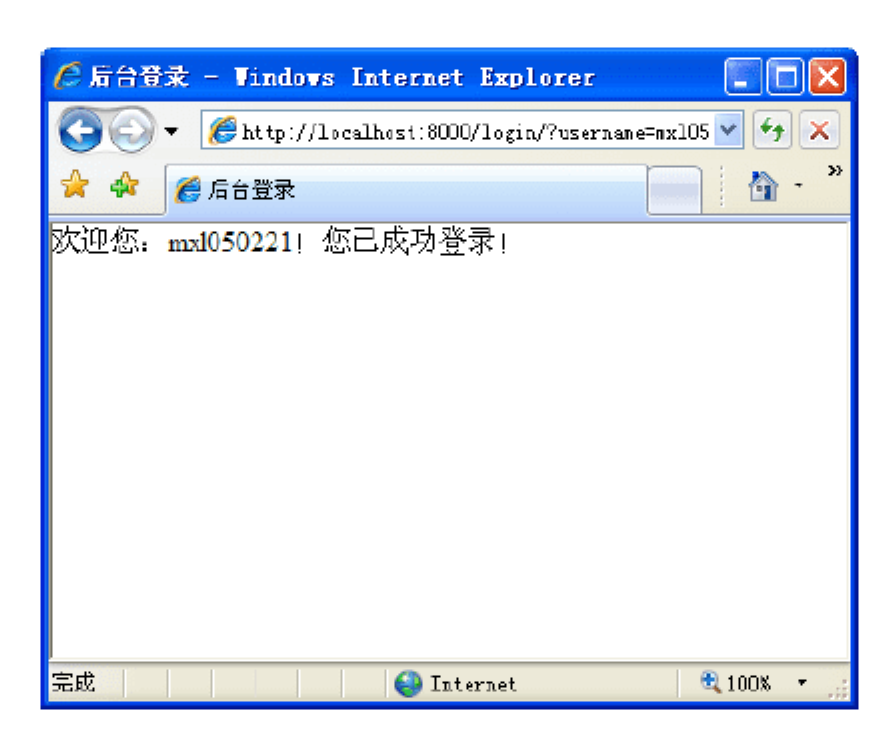


图 15-25　登录成功页面

15.5.4　实例描述

在这个网络飞速发展的年代，就连购物也达到了足不出户的地步。从网上购物之快捷众人皆知，只要轻轻单击几下鼠标，选择所要购买的商品，该商品就会进入自己的购物车，以待付款。下面使用 Django 框架中的 Session 对象来实现一个购物车。

15.5.5　实例应用

【例 15-2】使用 Django 框架中的 Session 实现购物车功能。

(1) 在 Django_Pro 项目中新建 Products 应用。打开 Windows 命令提示符，使用 cd 命令转到 Django_Pro 目录下，然后输入下面语句。

```
manage.py startapp Products
```

(2) 创建数据模型。打开 Products 目录下的 models.py 文件，在该文件中新建 Products 类，该类内容如下：

```
from django.db import models
class Products(models.Model):
    name = models.CharField('图书名称',max_length=20)
    publish = models.CharField('出版社',max_length=20)
    price = models.FloatField('定价',max_length=255)
    def __unicode__ (self):
        return '%s'%(self.name)
```

Products 类继承自 django.db 包中的 models 类。在 Products 类中定义 3 个属性来描述商品的相关信息。最后使用__unicode__()方法来将图书名称转换为 unicode 编码并返回。

(3) 启用 Session。修改 settings.py 文件中的 MIDDLEWARE_CLASSES 元组类型的变量，在该元组中添加下列语句。

```
'django.contrib.sessions.middleware.SessionMiddleware',
```

(4) 修改 settings.py 文件中的 INSTALLED_APPS 元组类型的变量，在该元组中加入两个元素。

```
'django.contrib.sessions',
'Django_Pro.Products',
```

(5) 再次打开 Windows 命令提示符，使用 cd 转到 Django_Pro 目录下，然后输入：

```
manage.py syncdb
```

按回车键，如果输出以下信息则表示创建成功。

```
Creating table Products_products
```

(6) 在 Django_Pro 目录下的 views.py 文件中新建 proList()方法。在该方法中使用 Products 类的 objects 对象中的 all()方法获取该类所对应的数据模型中的所有数据，并将数据列表传递到 pro_index.html 页面。proList()方法的内容如下：

```
from Django_Pro.Products.models import Products
def  proList (request):
    # 获取所有数据
    pros=Products.objects.all()
    return render_to_response('pro_index.html',{'pros':pros})
```

(7) 在 Django_Pro 项目根目录下的 urls.py 文件中配置匹配 URL，在 urlpatterns 变量中加入以下语句：

```
(r'^pro/$','Django_Pro.views.proList'),
```

这样配置之后，在浏览器的地址栏输入 http://localhost:8000/pro/，即可访问 views.py 文件中的 proList()方法，从而跳转至 pro_index.html 页面。

(8) 在项目的根目录下的 templates 模板目录下新建 pro_index.html 页面，在该页面中显示商品列表信息。当单击商品名称时，该商品加入购物车，同时将该商品的 ID 值作为参数传递到路径为/buy/所对应的 URL 中。pro_index.html 页面的代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
    <head>
        <title> 图书列表</title>
    </head>
    <body>
    {% if pros %}
        {% for pro in pros%}
            <a href='/buy/?proId={{ pro.id }}'><b>{{pro}}</b></a><br/><br/>
        {% endfor %}
    {% else %}
        暂无数据
```

```
    {% endif %}
    </body>
</html>
```

(9) 在根目录下的 urls.py 文件中配置如下的 URL 匹配路径：

```
(r'^buy/$','Django_Pro.views.buyPro'),
```

该语句表示，当访问 buy/路径时，将调用根目录下的 views.py 文件中的 buyPro()方法。

(10) 在根目录下的 views.py 文件中创建 buyPro()方法，在该方法中接收 pro_index.html 页面传递过来的 proId 参数，并判断该参数值是否大于 0，如果大于 0，则使用 Products 类的 objects 对象中的 get()方法根据 ID 值获取一个 Products 对象，然后将获取的该对象存入 Session 中。在这里，需要在模板页面中使用 request 对象(Django 在默认情况下不允许)，需要在 views.py 文件中导入 RequestContext 类，同时需要在 render_to_response()方法的第二个参数以 context_instance=RequestContext(request)语句将 request 对象添加至视图中。buyPro()方法的代码如下：

```
from django.template import RequestContext
def buyPro (request):
    # 获取商品编号
    proId=request.GET.get('proId',None)
    if proId > 0:
        # 根据商品编号获取特定的商品信息
        pro=Products.objects.get(id=proId)
        # 将商品信息存入 Session 中
        request.session['pro']=pro
        return render_to_response('showBuy.html',context_instance=
RequestContext(request))
    return render_to_response('showBuy.html')
```

(11) 在 settings.py 文件中配置 TEMPLATE_CONTEXT_PROCESSORS 元组类型的变量。具体配置如下：

```
TEMPLATE_CONTEXT_PROCESSORS = (
    "django.core.context_processors.auth",
    "django.core.context_processors.debug",
    "django.core.context_processors.i18n",
    "django.core.context_processors.media",
    "django.core.context_processors.request"
)
```

(12) 在 templates 目录下新建 showBuy.html 文件，在该文件中显示 Session 中保存的商品信息。showBuy.html 页面代码如下：

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
  <title> 购物车 </title>
  <style type="text/css">
  th{
      font-size:14px;
      background-color:#CCCCCC;
      border-color:#000000;
```

```
      height:30px;
  }
  td{
      font-size:12px;
      text-align:center;
      border-color:#000000;
      height:25px;
  }
  </style>
  </head>
  <body>
      <h2>购物清单</h2>
      <hr/>
      <table cellpadding="0" cellspacing="0" border="1"
style="border-color:#000000; border-width:thin;" width="100%">
          <tr>
              <th width="40%">图书名称</th><th width="40%">出版社</th><th>定价</th>
          </tr>
          <tr>
              <td>{{request.session.pro.name}}</td><td>{{request.session.pro.publish}}
</td><td>{{request.session.pro.price}} </td>
          </tr>
      </table>
  </body>
</html>
```

15.5.6　运行结果

打开 IE 浏览器，然后在地址栏输入 http://localhost:8000/pro/，按回车键，显示所有商品的名称，如图 15-26 所示。当单击某一件商品的名称时，该商品被投入购物车，同时显示该商品的详细信息，如图 15-27 所示。

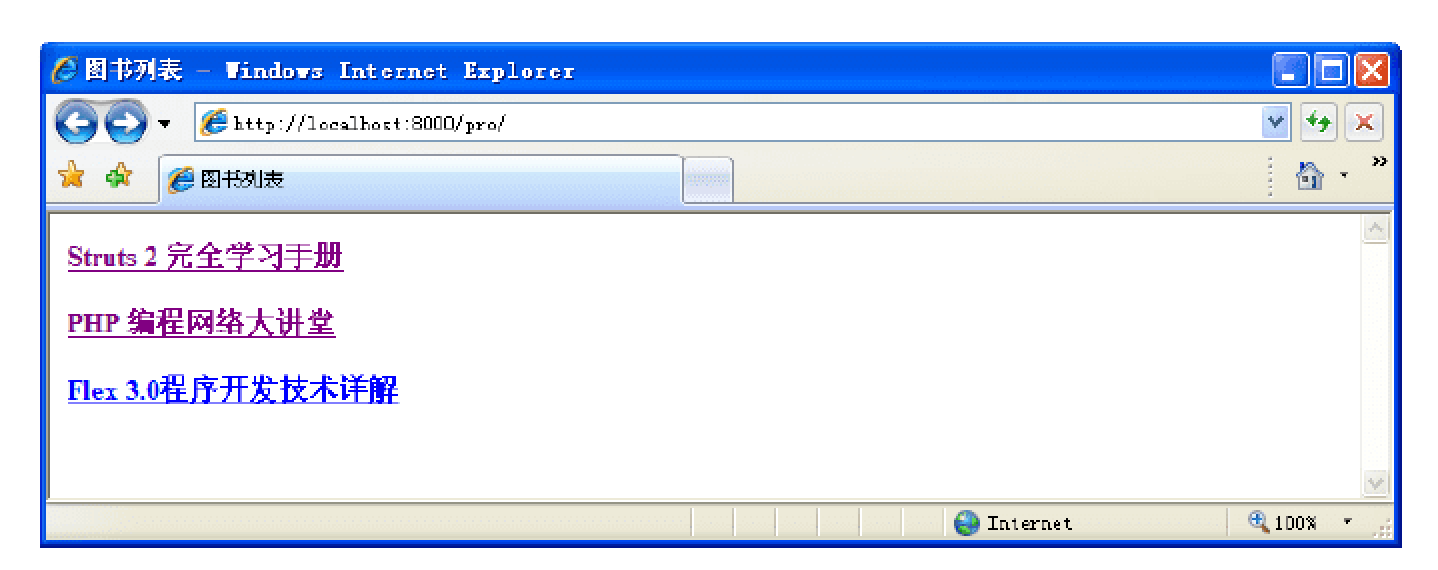


图 15-26　商品列表信息



图 15-27　购物车列表

15.5.7　实例分析



源码解析

在该案例中，将一个特定的 Products 对象存放于 Session 中，而非字符串。由于 Django 框架中的视图不允许使用 request 对象，因此需要使用 RequestContext()类的构造函数将 request 对象注册到 Django 框架的视图中，这样注册之后仍然不行，还需要在 settings.py 文件中配置 TEMPLATE_CONTEXT_PROCESSORS 变量，将 django.core.context_processors.request 加入到该变量所表示的元组中，这样配置之后即可在视图中使用{{request.session.pro.name}}语句来获取 Session 中 pro 所表示的 Products 类中的属性值。

15.6　常见问题解答

15.6.1　出现AttributeError: 'str' object has no attribute '_meta'错误



出现 AttributeError: 'str' object has no attribute '_meta'错误！
网络课堂：http://bbs.itzcn.com/thread-15823-1-1.html

需要在生成的数据模型中加入一个外键关联，下面是我的代码：

```
user = models.ForeignKey('User')
```

这样编辑之后，却出现 AttributeError: 'str' object has no attribute '_meta'的错误信息，这是怎么回事？

【解决办法】外键关联的只是一个表中的 ID，即一个数据模型中的 ID。应改写为下面的语句。

```
user = models.ForeignKey(User)
```

15.6.2　Django出现UnicodeEncodeError错误



Django 出现 UnicodeEncodeError 错误？
网络课堂：http://bbs.itzcn.com/thread-15824-1-1.html

原先使用 Python 2.4+Django 0.95 做的程序，现在把环境升级到 Python 2.5+Django 1.3，并且在把两个表做了外键关联的情况下，发现在后台添加数据时，会出现错误，错误信息如下：

```
UnicodeEncodeError: 'ascii' codec can't encode characters in position 0-1:
ordinal not in range(128)
```

【解决办法】Python 中有两种字符串，分别是一般字符串(每个字符用 8bites 表示)和 Unicode 字符串(每个字符用一个或者多个字节表示)，它们可以相互转换。从错误提示来看是由于 Python 遇到了编码的问题。也就是说，后台输入的数据在默认情况下是 ASCII 编码的，那么在存入数据库时，Python 便会报错，但是数据库中已经插入了该条数据记录。解决该问题的方法很简单，只需要将原有程序的数据模型代码中的方法由 def __str__(self)改为 def __unicode__(self)即可。__str__()是旧版本中采用的方法，在 Django 0.96 以上版本中，该方法已经被__unicode__()方法替换。这样修改之后，就解决了字符串传递时出错的问题，即统一编码为 UTF-8。

15.6.3　程序升级到Django 1.0 后遇到问题



程序升级到 Django 1.0 后遇到问题！
网络课堂：http://bbs.itzcn.com/thread-15825-1-1.html

从 Django 0.96 升级到 Django 1.0，程序在本地运行和调试一切正常，但发布到 Linux 服务器上却出现了 500(服务器内部错误)错误页面。查看日志文件，也看不到任何错误信息，如何解决？

【解决办法】将原来程序中的 from django.newforms.widgets import Widget 语句改为 from django.forms.widgets import Widge 语句，就解决了 500 错误。因为版本的不同，模块也有所不同。

15.7 习　　题

一、填空题

(1) 启动 Django 项目内置的开发服务器需要使用 manage.py 文件中的__________命令。

(2) 如果一个应用程序的模板目录为根目录下的 users 文件夹中的 template 目录，下面是 settings.py 文件中的一段代码，在画线处应填写__________。

```
TEMPLATE_DIRS = (
    '__________',
)
```

(3) 在 Django 应用中修改 settings.py 文件中的 INSTALLED_APPS 元组变量值，即加入__________元素值，才能在 Django 应用中访问后台管理系统页面。

二、选择题

(1) 创建一个名称为 blog 的项目，需要使用__________语句。

A. django-admin.py help startproject blog

B. django-admin.py startproject blog

C. django-admin.py startapp blog

D. manage.py startproject blog

(2) 当在地址栏输入 http://localhost:8000/mysite/时，需要调用 mysite 应用中的 site 子应用下 views.py 文件中的 mysite()方法。那么在 mysite 目录下的 urls.py 文件中应选择_________代码。

A.

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^mysite/','mysite.site.views.mysite'),
)
```

B.

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^mysite/',include('mysite.site.views.mysite')),
)
```

C.

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^mysite/','mysite.site.views'),
)
```

D.

```
from django.conf.urls.defaults import patterns, include, url
urlpatterns = patterns('',
    (r'^mysite/',include('mysite.site.views')),
)
```

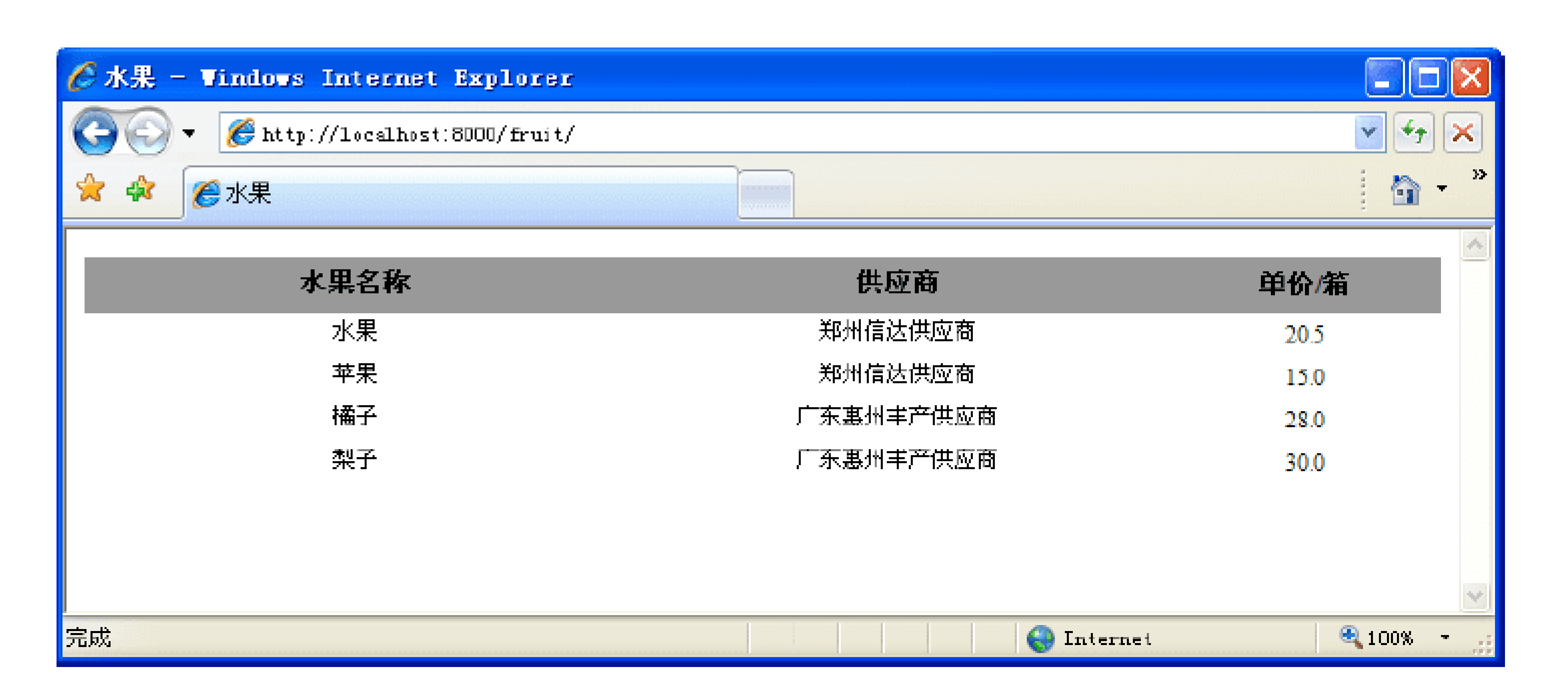
(3) 如果需要在 Django 应用的视图中使用 request 对象，则需要借助于__________类将 request 对象注册到视图中。

A. HttpResponse　　　　B. HttpResponseRedirect
C. render_to_response　　　　D. RequestContext

三、上机练习

上机练习：获取水果列表。

在 Django_Pro 目录下生成一个 Fruit 应用，并在该应用下的 models 模块中创建 Fruit 类，该类所对应的数据模型有 4 个字段，分别是 id、name、factruer 和 price，依次表示水果编号、水果名称、供应商和单价。在 Django_Pro 目录下的 urls.py 文件中配置该应用的 URL 匹配信息：当访问 fruit/路径时，程序调用根目录下的 views.py 文件中的 fruitList()方法，在该方法中获取水果列表信息后跳转至 fruit_list.html 模板页面，并将获取的水果列表作为参数传递过去。在该模板页面中判断传递过来的水果列表参数是否有值，如果有值，则循环遍历该列表，并将水果的详细信息输出到页面，否则显示“无数据”。运行结果如图 15-28 所示。



水果名称	供应商	单价/箱
水果	郑州信达供应商	20.5
苹果	郑州信达供应商	15.0
橘子	广东惠州丰产供应商	28.0
梨子	广东惠州丰产供应商	30.0

图 15-28　水果列表信息

附录　各章习题参考答案

第 1 章

一、填空题

(1) Guido van Rossum
(2) 三
(3) 解释性

二、选择题

(1) B
(2) D
(3) A

第 2 章

一、填空题

(1) #
(2) \
(3) 局部变量

二、选择题

(1) A
(2) A
(3) C

第 3 章

一、填空题

(1) True
(2) elif
(3) while

(4) for
(5) break
(6) continue
(7) pass

二、选择题

(1) D
(2) C
(3) B
(4) A

第 4 章

一、填空题

(1) def
(2) 0
(3) from MyModule import *
(4) __name__

二、选择题

(1) D
(2) B
(3) D
(4) C

第 5 章

一、填空题

(1) d 'de' 'abcde' 'def'
(2) clear() keys() values()
(3) 2

二、选择题

(1) C
(2) A
(3) D

第 6 章

一、填空题

(1) %s
(2) join()
(3) endswith()
(4) strftime()
(5) %T

二、选择题

(1) C
(2) B
(3) C

第 7 章

一、填空题

(1) __
(2) @staticmethod
(3) @classmethod
(4) person.began=ending
(5) Issubclass
(6) __slots__
(7) __getattribute__()

二、选择题

(1) A
(2) B
(3) C
(4) D

第 8 章

一、填空题

(1) read()

(2) 列表内容
(3) shutil
(4) remove()
(5) mkdir()

二、选择题

(1) B
(2) D
(3) D

第 9 章

一、填空题

(1) StandardError
(2) assert
(3) ZeroDivisionError

二、选择题

(1) C
(2) A
(3) D

第 10 章

一、填空题

(1) shelve
(2) shelve
(3) connect
(4) execute
(5) Fetchall()

二、选择题

(1) A
(2) C
(3) D

第 11 章

一、填空题

(1) socket
(2) loop()
(3) SocketServer

二、选择题

(1) B
(2) C
(3) C

第 12 章

一、填空题

(1) urlparse()
(2) urlparse
(3) urlopen
(4) httplib
(5) handle_data

二、选择题

(1) A
(2) B
(3) C
(4) D

第 13 章

一、填空题

(1) >
(2) CDATA
(3) startDocument()
(4) startElement
(5) DTDHandler

(6) DOMImplementation
(7) implementation
(8) nodeType

二、选择题

(1) A
(2) B
(3) C
(4) D
(5) B

第 14 章

一、填空题

(1) OnInit()
(2) CreateToolBar()
(3) AppendMenu

二、选择题

(1) B
(2) A
(3) C

第 15 章

一、填空题

(1) runserver
(2) ./users/template
(3) django.contrib.admin

二、选择题

(1) B
(2) A
(3) D